Using Contracted Solution Graphs for Solving Reconfiguration Problems*

Paul Bonsma**[1] and Daniël Paulusma***[2]

[1] Saxion University of Applied Sciences, Enschede, The Netherlands, p.s.bonsma@saxion.nl
[2] Durham University, UK, daniel.paulusma@durham.ac.uk

Abstract. We introduce in a general setting a dynamic programming method for solving reconfiguration problems. Our method is based on *contracted solution graphs*, which are obtained from solution graphs by performing an appropriate series of edge contractions that decrease the graph size without losing any critical information needed to solve the reconfiguration problem under consideration. Our general framework captures the approach behind known reconfiguration results of Bonsma (2017) and Hatanaka, Ito and Zhou (2015). As a third example, we apply the method to the following well-studied problem: given two k-colorings α and β of a graph G, can α be modified into β by recoloring one vertex of G at a time, while maintaining a k-coloring throughout? This problem is known to be PSPACE-hard even for bipartite planar graphs and $k = 4$. By applying our method in combination with a thorough exploitation of the graph structure we obtain a polynomial-time algorithm for $(k-2)$-connected chordal graphs.

Keywords. reconfiguration, contraction, dynamic programming, graph coloring.

1 Introduction

Solving a given instance of an NP-hard search problem means that we need to explore an exponentially large solution space. In order to get more insight into the solution space, it is a natural question to check how "close"' one solution for a particular instance is to another solution of that instance. Doing so could, for instance, be potentially interesting for improving the performance of corresponding heuristics [18]. Searching the solution space by making small "feasible" moves also turned out to be useful when analyzing randomized algorithms for sampling and counting k-colorings of a graph or when analyzing cases of Glauber dynamics in statistical physics (see Section 5 of the survey of van den Heuvel [23]). Solution spaces in practical problems, such as stacking problems arising in storage planning [32], have been explored in a similar matter.

The above situation can be modeled as follows. A *solution graph concept* $\mathcal{S}$ is obtained by defining a set of *instances*, *solutions* for these instances, and a (symmetric) *adjacency relation* between pairs of solutions. For every instance G of the problem, this gives a *solution graph* $\mathcal{S}(G)$, also called a *reconfiguration graph*, which has as node set all solutions of G, with edges as defined by some given adjacency relation (if G has no solutions, then $\mathcal{S}(G)$ is the empty graph). The adjacency relation usually represents a smallest possible change (*reconfiguration move*) between two solutions of the same instance. For example, the well-known *k-Color Graph concept* $\mathcal{C}_k$, related to the k-Coloring search problem, is defined as follows: instances are graphs G, and solutions are (proper) k-colorings of G. Two k-colorings are adjacent nodes in the reconfiguration graph $\mathcal{C}_k(G)$ if and only if they differ in exactly

* An extended abstract of this paper appeared in the proceedings of MFCS 2016 [11].
** Supported by the European Community's Seventh Framework Programme (FP7/2007-2013), grant agreement n° 317662.
*** Supported by EPSRC Grant EP/K025090/1.

one vertex of G. In general there may be more than one natural way to define the adjacency relation.

Solution graphs and their properties have been studied very intensively over the last couple of years for a variety of search problems, such as k-Coloring [2–4, 8, 13–15, 17, 29], Satisfiability [18, 34], Independent Set [6, 9, 30], Shortest Path [7, 5, 30], List Coloring [20], List Edge Coloring [26, 28], $L(2,1)$-Labeling [27], H-Coloring [41] and Subset Sum [25]; see also the surveys [23, 39]. The study of such solution graphs is commonly called *reconfiguration*.

The area of reconfiguration is fast growing, and both algorithmic and combinatorial questions have been considered. For instance, what is the diameter of $\mathcal{S}(G)$ (in terms of the size of the instance G) or if $\mathcal{S}(G)$ is not connected, what is the diameter of its (connected) components? In particular, is the diameter always polynomially bounded or not? This led to the introduction of the $\mathcal{S}$-Connectivity problem, which is that of deciding whether the solution graph $\mathcal{S}(G)$ of a given instance G is connected. We consider the following related problem, which is also a central problem in the area of reconfiguration:

$\mathcal{S}$-Reachability
Instance: an instance G with two solutions α and β.
Question: is there a path from α to β in $\mathcal{S}(G)$?

The $\mathcal{S}$-Reachability problem is sometimes called the *α-β-path* problem for $\mathcal{S}$ [23]. Our example problem $\mathcal{C}_k$-Reachability is also known as the k-Color Path problem [15].

$\mathcal{C}_k$-Reachability
Instance: an instance G with two k-colorings α and β.
Question: is there a path from α to β in $\mathcal{C}_k(G)$?

It is known that $\mathcal{S}$-Reachability is PSPACE-complete for most of the aforementioned solution graph concepts even for special classes of instances [8, 19, 24, 37, 42, 43]. For instance, $\mathcal{C}_k$-Reachability is PSPACE-complete even if $k = 4$ and instances are restricted to planar bipartite graphs [8]. This explains that efficient algorithms are only known for very restricted classes of instances. Hence, there is still a need for developing general algorithmic techniques for solving these problems in practice, and for sharpening the boundary between tractable and computationally hard instance classes.

One important algorithmic technique is dynamic programming (DP). There are only relatively few successful examples of nontrivial dynamic programming algorithms for solving $\mathcal{S}$-Reachability problems. The reason for this is that many well-studied $\mathcal{S}$-Reachability problems (including $\mathcal{C}_k$-Reachability for an appropriate constant k) are PSPACE-complete even for graphs of bounded bandwidth [37, 42], and therefore also for graphs of bounded treewidth. In fact, the PSPACE-completeness results from [37, 42] hold even for planar graphs of bounded bandwidth and low maximum degree [43].

One way to cope with the above problem is to restrict the problem even further. For instance, in a number of recent papers [10, 22, 29, 30, 34–36] the *length-bounded* version of the $\mathcal{S}$-Reachability problem was studied. This is the problem of finding a path of length at most ℓ in the solution graph between two given solutions. Taking the length ℓ of a path between two solutions as a natural parameter, a particular aim of these papers was to determine fixed-parameter tractability. For instance, although $\mathcal{C}_k$-Reachability is PSPACE-complete for $k \geq$

4, the length-bounded version is FPT when parameterized by the length ℓ [10, 29] (for $k \leq 3$, the length-bounded version is even polynomial-time solvable [29]). In this restricted context, dynamic programming algorithms over tree decompositions for reconfiguration problems are more common. For instance, in [37] FPT algorithms are given for various length-bounded reachability problems, parameterized by both the treewidth and the length ℓ. To give another example, in [33] FPT algorithms are given for the reachability versions of different token reconfiguration problems for graphs of bounded degeneracy (and thus for bounded treewidth), when parameterized by the number of tokens.

1.1 Aims and Methodology

We aim to solve the (original) $\mathcal{S}$-Reachability problem via an algorithm that uses a generally applicable DP method based on *contracted solution graphs*. Due to the PSPACE-completeness of $\mathcal{S}$-Reachability, such an algorithm does not terminate in polynomial time for all instances. Hence we aim to identify restricted instance classes for which we do obtain a polynomial running time, as illustrated by a new application explained in Section 1.2 and two known examples [7, 20] explained below.

Bonsma [7] introduced the DP method based on contracted solution graphs to obtain an efficient algorithm for Shortest-Path-Reachability restricted to planar graphs. Hatanaka, Ito and Zhou [20] used this DP method for proving that List-Coloring-Reachability is polynomial-time solvable for caterpillars (trees obtained from paths possibly by adding additional vertices that have exactly one neighbor, which belongs to the path). In both papers, contracted solution graphs are called *encodings*. To be more precise, in [20] dynamic programming was done over a *path decomposition* of the given caterpillar. In [7], a layer-based decomposition of the graph was used, which can also be viewed as a path decomposition. In our paper we focus on the more general *tree decompositions* instead (which requires us to introduce a join rule). We will generalize the ideas of [7, 20] to a unified DP method and illustrate the method by giving a new application.

We now sketch our method and refer to Section 3 for a detailed description. In dynamic programming one first computes the required information for parts of an instance G. One combines/propagates this to compute the same information for ever larger parts of the instance, until the desired information is known for G entirely. In our case, G can be any relational structure on a ground set, such as (directed) graphs, hypergraphs, satisfiability formulas, or constraint satisfaction problems in general (see e.g. [10]). The order in which the information can be computed or the order in which parts must be considered is given by a *decomposition* of G. For a processed part H of G, the elements of the ground set that are in H and that have incidences with the unexplored part are called *terminals*. Reconfiguration moves in H that do not involve terminals are often irrelevant. We capture the information that is relevant by the notion of a *terminal projection*. These projections assign labels to solutions, yielding so-called *label components*, which are maximally connected subgraphs of $\mathcal{S}(H)$ induced by sets of solutions that all have the same label. A *contracted solution graph* is obtained from $\mathcal{S}(H)$ by contracting the label components into single vertices. Dynamic programming rules for a given decomposition of G describe how to compute new (larger) contracted solution graphs from smaller ones.

In Section 4 we illustrate our method by giving dynamic programming rules for the $\mathcal{C}_k$-Reachability problem that can be used if a tree decomposition of the graph is given. Recall that similar dynamic programming rules have been for other reconfiguration problems [7,

20] when a path decomposition is given. Our rules solve the $\mathcal{C}_k$-Reachability problem correctly for every graph G and can also be used directly for List-Coloring-Reachability and thus generalize the rules of [20]. Nevertheless, the algorithm is only *efficient* when the contracted solution graphs stay small enough (that is, polynomially bounded). As indicated by the aforementioned PSPACE-hardness of $\mathcal{C}_k$-Reachability, this is not always the case. To make this explicitly clear, in Section 5, we apply the DP rules on a specific example, which shows that the size of the contracted solution graphs can indeed grow exponentially, even for 2-connected 4-colorable unit interval graphs.

1.2 Application

In Section 6 we apply the DP method to show that, for all $k \geq 3$, $\mathcal{C}_k$-Reachability is polynomial-time solvable for $(k-2)$-connected chordal graphs. Chordal graphs form a well-studied graph class; see e.g. [12] for more information. As unit interval graphs are chordal, the example given in Section 5 implies that we need to exploit the structure of chordal graphs further in combination with applying the DP method. The idea is to show that it suffices to compute the contracted solution graphs only partly. In order to do this we introduce the new notion of injective neighbourhood property of contracted solution graphs for $\mathcal{C}_k$-Reachability, which helps us to characterize contracted solution graphs if the original graph G is chordal and $(k-2)$-connected.

As the proof for the PSPACE-completeness result for bipartite graphs from [8] can be easily modified to hold for $(k-2)$-connected bipartite graphs[3], our result for $(k-2)$-connected chordal graphs cannot be extended to $(k-2)$-connected perfect graphs. Our result cannot be extended to all chordal graphs either: recently, Hatanaka, Ito and Zhou [21] solved an open problem posed in the conference version of our paper [11] by proving that $\mathcal{C}_k$-Reachability is PSPACE-complete for chordal graphs if k is a sufficiently large constant. We note that in contrast, $\mathcal{C}_k$-Connectivity is polynomial-time solvable on chordal graphs. This is due to a more general result of Bonamy et al. [4], which implies that for a chordal graph G, $\mathcal{C}_k(G)$ is connected if and only if G has no clique with more than $k-1$ vertices.

Our result on $\mathcal{C}_k$-Reachability on $(k-2)$-connected chordal graphs is the first time that dynamic programming over tree decompositions is used to solve the general version of a PSPACE-complete reachability problem in polynomial time for a graph class strictly broader than trees. As reachability problems become quickly PSPACE-complete, we can only hope to obtain polynomial-time algorithms for rather restricted graph classes. Nevertheless we believe that there are a number of interesting open problems in this direction, for which our method could be useful (see Section 7). We also remark that the true strength of our method is not always revealed when using the viewpoint of worst-case algorithm analysis. For instance, we observed from some initial experiments using randomly generated k-colorable chordal or interval graphs that the method performs well on most instances, despite the fact that

[3] In [8], it is first shown that the *list-coloring* version of the $\mathcal{C}_k$-Reachability problem is PSPACE-complete for connected planar bipartite 4-colorable graphs. In the list-coloring version every vertex is assigned a list of two or three allowed colors. In the next step of the transformation, to the $\mathcal{C}_k$-Reachability problem, the list-coloring constraints are simulated by making every vertex adjacent to appropriately colored vertices of a small bipartite graph with a frozen 4-coloring. A *frozen k-coloring* is a k-coloring where every vertex is adjacent to vertices of each other color, so no vertex can be recolored. Since we do not care about planarity, we can instead simply take one highly connected bipartite graph with a frozen k-coloring and link all vertices of the first graph to the appropriate vertices of this new graph, to enforce the list-color constraints, while maintaining bipartiteness and ensuring $(k-2)$-connectedness.

specialized examples can be constructed that exhibit exponential growth. Discussing this is beyond the scope of the current paper. However, being able to use our method for performing experiments with a goal to further refining it was one of the reasons why we choose to present it in full generality.

1.3 Paper Organization

To briefly summarize the organization of our paper, Section 2 contains basic notation and terminology used throughout the paper. In Section 3 we give a full description of the dynamic programming method of contracted solution graphs. This method relies on rules for dynamic programming. In Section 4 we illustrate our method by giving such rules, based on tree decompositions, for the $\mathcal{C}_k$-Reachability problem. Before we apply the method of Section 3 with the rules of Section 4 on the $\mathcal{C}_k$-Reachability problem for $(k-2)$-connected chordal graphs in Section 6, we first give some examples of the use of these rules in Section 5. The purpose of the latter section is also to illustrate that contracted solution graphs may have exponential size. We conclude our paper with giving directions for future work in Section 7.

2 Preliminaries

We consider finite undirected graphs that have no multi-edges and no loops. Below we define some basic terminology. In particular we give some coloring terminology, as we need such terminology throughout the paper. We refer to the textbook of Diestel [16] for any undefined terms.

For a connected graph G, a *vertex cut set* is a set $S \subseteq V(G)$ such that $G-S$ is disconnected. Vertices in different components of $G-S$ are said to be *separated* by S. For $k \geq 1$, a (connected) graph G is *k-connected* if $|V(G)| \geq k+1$ and every vertex cut set S has $|S| \geq k$. The *contraction* of an edge uv of a graph G replaces u and v by a new vertex made adjacent to precisely those vertices that were adjacent to u or v in G (this does not create any multi-edges or loops). A graph is *chordal* if it has no induced cycle of length greater than 3.

Let G be a graph. A *k-color assignment* of G is a function $\alpha : V(G) \to \{1, \ldots, k\}$. For $v \in V(G)$, $\alpha(v)$ is called the *color* of v. A k-color assignment α is a (proper) *k-coloring* if $\alpha(u) \neq \alpha(v)$ for every edge $uv \in E(G)$ (note that a k-coloring may use less than k colors). A *coloring* of G is a k-coloring for some value of k. If α and β are colorings of G and a subgraph H of G, respectively, such that $\alpha|_{V(H)} = \beta$ (that is, α and β coincide on $V(H)$) then α and β are said to be *compatible*.

For an integer k, the *k-color graph* $\mathcal{C}_k(G)$ has as nodes all (proper) k-colorings of G, such that two colorings are adjacent if and only if they differ on one vertex. Note that two k-colorings of G that can be obtained from each other by a permutation of the colors correspond to distinct nodes in $\mathcal{C}_k(G)$. A *walk* from u to v in G is a sequence of vertices $v_0, \ldots, v_k$ with $u = v_0$, $v = v_k$, such that for all $i < k$, $v_i v_{i+1} \in E(G)$. A *pseudowalk* from u to v is a sequence of vertices $v_0, \ldots, v_k$ with $u = v_0$, $v = v_k$, such that for all $i < k$, either $v_i = v_{i+1}$, or $v_i v_{i+1} \in E(G)$. A *recoloring sequence* from a k-coloring α of G to a k-coloring β of G is a pseudowalk from α to β in $\mathcal{C}_k(G)$ (in this definition we use the notion of a pseudowalk as we need this in the context of contracted solution graphs).

A *labeled graph* is a pair (G, ℓ) where $G = (V, E)$ is a graph and $\ell : V \to X$ for some set X is a vertex labeling (which may assign the same label to different vertices); we refer to Section 3 for an example. A *label preserving isomorphism* between two labeled graphs

(G_1, ℓ_1) and (G_2, ℓ_2) is an isomorphism $\phi : V(G_1) \to V(G_2)$, such that $\ell_1(v) = \ell_2(\phi(v))$ for all $v \in V(G_1)$. Informally, two labeled graphs (G_1, ℓ_1) and (G_2, ℓ_2) are the same if there exists a label preserving isomorphism between them.

3 The Method of Contracted Solution Graphs

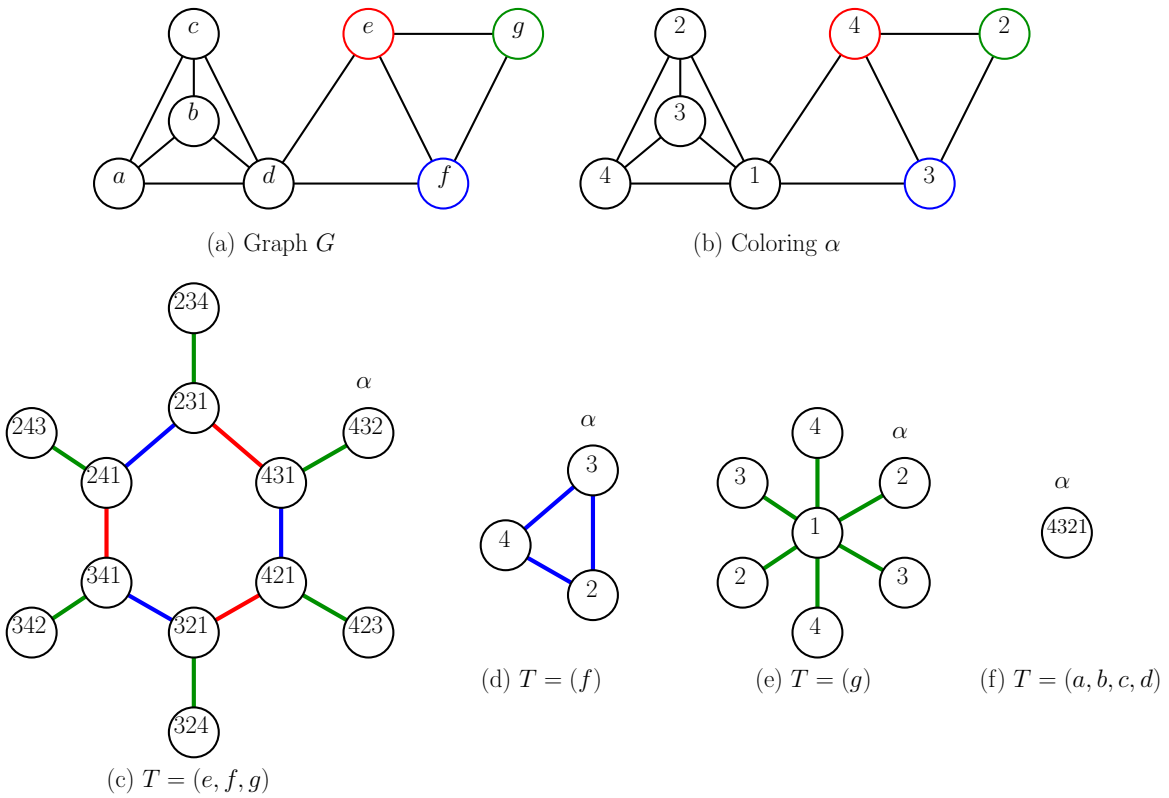
In this section we define the concept of *contracted solution graphs* (CSGs) for reconfiguration problems in general. Consider a solution graph concept $\mathcal{S}$, which for every instance G of $\mathcal{S}$ defines a solution graph that is denoted by $\mathcal{S}(G)$. A *terminal projection* for $\mathcal{S}$ is a function p that assigns a *label* to each tuple (G, T, γ) consisting of an instance G of $\mathcal{S}$, a set T of *terminals* for G and a solution γ for G. Terminal projections are used to decide which nodes are "equivalent" and can be contracted. In our example and in previous examples in the literature [7, 20] G is always a graph, and T is a subset of its vertices. A terminal projection p can be seen as a node labeling for the solution graph $\mathcal{S}(G)$. So, for every instance G of $\mathcal{S}$, every choice of terminals T may give a different node labeling for the solution graph $\mathcal{S}(G)$. If G and T are clear from the context, we write $p(\gamma)$ to denote the label of a node γ of $\mathcal{S}(G)$.

Example 1. Consider the k-color graph concept $\mathcal{C}_k$. Let G be a graph. We can define a terminal projection p as follows. Let T be a subset of $V(G)$. The nodes of $\mathcal{C}_k(G)$ are k-colorings and we give each node as label its restriction to T, that is, for every k-coloring γ of G, we set $p(\gamma) = p(G, T, \gamma) = \gamma|_T$. Note that $\gamma|_T$ is a k-coloring of $G[T]$.

Let p be a terminal projection for a solution graph concept $\mathcal{S}$. For an instance G of $\mathcal{S}$ and a terminal set T, a *label component* C of $\mathcal{S}(G)$ is a maximal set of nodes γ that all have the same label $p(\gamma)$ and that induce a connected subgraph of $\mathcal{S}(G)$. It is easy to see that every solution γ of G is part of exactly one label component, or in other words: the label components partition the node set of $\mathcal{S}(G)$. The *contracted solution graph (CSG)* $\mathcal{S}^c(G, T)$ is a labeled graph that has a node set that corresponds bijectively to the set of label components of G. For a node x of $\mathcal{S}^c(G, T)$, we denote by S_x the corresponding label component. Two distinct nodes x_1 and x_2 of $\mathcal{S}^c(G, T)$ are adjacent if and only if there exist solutions $\gamma_1 \in S_{x_1}$ and $\gamma_2 \in S_{x_2}$ such that γ_1 and γ_2 are adjacent in $\mathcal{S}(G)$. We define a label function ℓ^* for nodes of $\mathcal{S}^c(G, T)$ to denote the corresponding label in $\mathcal{S}(G)$. More precisely: for a node x of $\mathcal{S}^c(G, T)$, the label $\ell^*(x)$ is chosen such that $\ell^*(x) = p(\gamma)$ for all $\gamma \in S_x$. Note that the contracted solution graph $\mathcal{S}^c(G, T)$ can also be obtained from $\mathcal{S}(G)$ by contracting all label components into single nodes and choosing node labels appropriately.

Example 2. Figure 1(c) shows one component of $\mathcal{C}_4(G)$ for the (4-colorable) graph G from Figure 1(a). This is the component that contains all colorings of G whose vertices a, b, c, d are colored with colors $4, 3, 2, 1$, respectively (note that it is not possible to recolor any of these four vertices, as one may recolor only one vertex at a time). So in Figure 1(c) the colors of the vertices a, b, c, d are omitted in the node labels, which only indicate the colors of e, f, g, in this order. For terminal set $T = \{f\}$, this component contains three label components (of equal size), and contracting them yields the CSG $\mathcal{C}_4^c(G, \{f\})$ shown in Figure 1(d). For $T = \{g\}$, there are seven label components, and the corresponding CSG $\mathcal{C}_4^c(G, \{g\})$ is shown in Figure 1(e). Note that $\mathcal{C}_4^c(G, \{g\})$ contains different nodes with the same label.

We stress that the CSG $\mathcal{S}^c(G, T)$ is a labeled graph that includes the label function ℓ^* defined above. However, to keep its size reasonable, the CSG itself does not include the solution sets S_x for each node that were used to define it. For proving the correctness of dynamic



(a) Graph G (b) Coloring α

(c) $T = (e, f, g)$ (d) $T = (f)$ (e) $T = (g)$ (f) $T = (a, b, c, d)$

Fig. 1. (a) A 4-colorable chordal graph G with $V(G) = \{a, b, c, d, e, f, g\}$. (b) a 4-coloring α, and one component of the CSGs of G for four different terminal sets T: (c) $\mathcal{C}_4^c(G, \{e, f, g\})$, (d) $\mathcal{C}_4^c(G, \{f\})$, (e) $\mathcal{C}_4^c(G, \{g\})$ and (f) $\mathcal{C}_4^c(G, \{a, b, c, d\})$. The $G[T]$-colorings in the node labels are given as sequences of colors, for the (ordered version of) T as indicated below each CSG. The edge colors in examples (c), (d) and (e) correspond to the vertex colors in (b) and indicate in which vertex of G (namely, e, f or g) two adjacent colorings differ. Example (d) is constructed from example (c) by contracting an edge between two nodes with labels xyz and $x'y'z'$ (where x, y, z belong to $\{1, 2, 3, 4\}$) if and only if $y = y'$, eventually resulting in a node with label y. A similar claim holds for example (e). Example (c) can also be seen as the component of $\mathcal{C}_4(G)$ where vertices a, b, c, d receive colors $4, 3, 2, 1$.

programming rules for CSGs the following alternative characterization of CSGs is useful; note that the sets S_x correspond exactly to the label components.

Lemma 1 *Consider an instance G of a solution graph concept $\mathcal{S}$, terminal set T and terminal projection p. Let (H, ℓ) be a labeled graph. Then $(H, \ell) = \mathcal{S}^c(G, T)$ if and only if one can define (nonempty) sets of solutions S_x for each node $x \in V(H)$ such that the following properties hold:*

1. *$\{S_x \mid x \in V(H)\}$ is a partition of the nodes of $\mathcal{S}(G)$ (the solutions of G).*
2. *For every $x \in V(H)$ and every solution $\gamma \in S_x$: $p(G, T, \gamma) = \ell(x)$.*
3. *For every edge $xy \in E(H)$: $\ell(x) \neq \ell(y)$.*
4. *For every $x \in V(H)$: S_x induces a connected subgraph of $\mathcal{S}(G)$.*
5. *For every pair of distinct nodes $x, y \in V(H)$: $xy \in E(H)$ if and only if there exist solutions $\alpha \in S_x$ and $\beta \in S_y$ such that α and β are adjacent in $\mathcal{S}(G)$.*

Proof. $\Rightarrow$: We choose the sets S_x to be the label components, as chosen in the above definition of $\mathcal{S}^c(G, T)$. Then the Properties (1), (2), (4) and (5) follow immediately from the definitions. For Property (3), we use the fact that label components are *maximal* connected node sets with the same label, together with Properties (2) and (5).

$\Leftarrow$: Let (H, ℓ) be a labeled graph for which solution sets S_x can be defined such that the five properties hold. Consider a node $x \in V(H)$. Properties (4) and (2) show that all solutions in S_x are part of the same label component; denote this label component by C. Note that for all $\alpha \in S_x$, $p(G, T, \alpha) = \ell(x)$ due to Property (2).

In order to show that in fact $S_x = C$, let $\gamma \notin S_x$ be adjacent to some solution in S_x. We must show that $p(G, T, \gamma) \neq \ell(x)$. By Property (1), γ belongs to a set S_y for some $y \neq x$. By Property (5), we obtain $xy \in E(C)$. Then, by Property (3), we find that $\ell(y) \neq \ell(x)$. Hence $p(G, T, \gamma) = \ell(y) \neq \ell(x)$ due to Property (2).

Because S_x induces a label component for every x, there exists a bijection ϕ between the nodes of H and the label components of $\mathcal{S}(G)$ (This is a bijection because of Property (1)). This yields a bijection ϕ' between the nodes of H and the nodes of $\mathcal{S}^c(G, T)$, which is label preserving by Property (2) and an isomorphism by Property (5) and the definition of $\mathcal{S}^c(G, T)$. Hence $(H, \ell) = \mathcal{S}^c(G, T)$.[4] □

A mapping S that assigns solution sets (or label components) S_x to each node x of $\mathcal{S}^c(G, T)$ that satisfies the properties given in Lemma 1 is called a *certificate* for $\mathcal{S}^c(G, T)$. Given such a certificate S and a solution γ for G, we define the *γ-node* of $\mathcal{S}^c(G, T)$ with respect to S to be the node x with $\gamma \in S_x$. For readability, we will not always explicitly mention this certificate when talking about γ-nodes in $\mathcal{S}^c(G, T)$ (except in Lemma 2 below), but the reader should keep the following convention in mind: when γ-nodes are identified in $\mathcal{S}^c(G, T)$ for multiple solutions γ, *these are all chosen with respect to the same certificate.*

Example 3. In Figures 1(c)–(f), the α-node for the coloring α shown in Figure 1(b) is marked. In particular consider $\mathcal{C}_4^c(G, \{g\})$ in Figure 1(e). Since the certificate for $\mathcal{C}_4^c(G, \{g\})$ is not actually indicated in the figure, the other leaf with label 2 can also be chosen as the α-node (considering the nontrivial label-preserving automorphisms of the graph). Similarly, if we

[4] Since node names are irrelevant, we will simply write $(H, \ell) = \mathcal{S}^c(G, T)$ to denote that there is a label preserving isomorphism between the two. More formally, $\mathcal{S}^c(G, T)$ can be seen as a class of labeled graphs that are equivalent under labeled isomorphisms.

choose a coloring β that coincides with α except on nodes e and f, where we choose $\beta(e) = 3$ and $\beta(f) = 4$, then the same two leaves (the ones with label 2) of $\mathcal{C}_4^c(G, \{g\})$ can be chosen as the β-node. Nevertheless, if both an α-node and β-node are marked, then this will only be correct according to the above convention when they are distinct! [5]

The main purpose of our definitions is the following key lemma.

Lemma 2 *Let (G, T) be an instance of a solution graph concept $\mathcal{S}$. Let $\mathcal{S}^c(G, T)$ be the contracted solution graph for some terminal projection p. Let α and β be two solutions and let x and y be the α-node resp. β-node with respect to some certificate S. Then there is a path from α to β in $\mathcal{S}(G)$ if and only if there is a path from x to y in $\mathcal{S}^c(G, T)$.*

Proof. First suppose that there exists a path $\gamma_0, \ldots, \gamma_k$ from α to β in $\mathcal{S}(G)$. Replace every solution γ_i in this sequence by the node v of $\mathcal{S}^c(G, T)$ with $\gamma_i \in S_v$. By definition, the resulting node sequence starts in x, and terminates in y. By Lemma 1 Property 5, consecutive nodes in this sequence are the same or adjacent, so this sequence is a pseudowalk from x to y. This immediately yields a path from x to y.

For the other direction, consider a path $v_0, \ldots, v_k$ from x to y in $\mathcal{S}^c(G, T)$. For every node v_i, S_{v_i} induces a connected subgraph of $\mathcal{S}(G)$ (Lemma 1 Property 4). For any two consecutive nodes v_i and v_{i+1}, there exist solutions $\gamma \in S_{v_i}$ and $\gamma' \in S_{v_{i+1}}$ that are adjacent in $\mathcal{S}(G)$ (Lemma 1 Property 5). Clearly, $\alpha \in S_{v_0}$ and $\beta \in S_{v_k}$. Combining these facts yields a path from α to β in $\mathcal{S}(G)$. □

Lemma 2 implies that for a solution graph concept $\mathcal{S}$ and *any* terminal projection p and terminal set T, we can decide $\mathcal{S}$-Connectivity if we know $\mathcal{S}^c(G, T)$ (the answer is YES if and only if $\mathcal{S}^c(G, T)$ is connected) and the $\mathcal{S}$-Reachability problem if we know $\mathcal{S}^c(G, T)$ and the α-node and the β-node (the answer is YES if and only if these two nodes are in the same component). However, for obtaining an *efficient* algorithm using this strategy, we must throw away enough irrelevant information to ensure that $\mathcal{S}^c(G, T)$ will be significantly smaller than $\mathcal{S}(G)$, yet maintain enough information to ensure the efficient computation of $\mathcal{S}^c(G, T)$, without first constructing $\mathcal{S}(G)$. Our strategy for doing this is to use dynamic programming to compute $\mathcal{S}^c(H, T')$ for ever larger subgraphs H of G, while ensuring that all of the CSGs stay small throughout the process. The remainder of this paper shows a successful example of this strategy.

4 Dynamic Programming over Tree Decompositions

The following terminology is based on widely used techniques for dynamic programming over tree decompositions; see Section 6 and [1, 31, 38] for further information.

A *terminal graph* (G, T) is a graph G together with a vertex set $T \subseteq V(G)$, whose vertices are called the *terminals*. If $T = V(G)$, then (G, T) is called a *leaf*. If $v \in T$, then we say that the new terminal graph $(G, T \setminus \{v\})$ is obtained from (G, T) by *forgetting* v (or *using a forget operation*). If $T \neq V(G)$, $v \in T$ and $N(v) \subseteq T$ then we say that (G, T) can be obtained from $(G - v, T \setminus \{v\})$ by *introducing* v (or *using an introduce operation*). Note that for a terminal

[5] It is possible for the given example to choose two solutions α and β, and correctly mark an α-node x with respect to a one certificate S^1, and a β-node y with respect to another certificate S^2, such that α and β are in different components of $\mathcal{S}(G)$, but x and y are in the same component of $\mathcal{S}^c(G, T)$. This is clearly not desirable; see Lemma 2.

graph (G', T') with $T' \neq \emptyset$, different graphs can be obtained from (G', T') by introducing a vertex v, whereas forgetting a terminal always yields a unique result. The condition that each neighbor of the new vertex v must be in T is necessary, as we will see at several places in our proofs. We say that (G, T) is the *join of* (G_1, T) *and* (G_2, T) if

- G_1 and G_2 are induced subgraphs of G,
- $V(G_1) \cap V(G_2) = T$ and $V(G_1) \cup V(G_2) = V(G)$,
- $V(G_1) \neq T$ and $V(G_2) \neq T$, and
- for every $uv \in E(G)$, it holds that $uv \in E(G_1)$ or $uv \in E(G_2)$.

Before we present the dynamic programming rules for recoloring, we first give some background and an overview of how these operations are commonly used in the context of dynamic programming over tree decompositions (see e.g. [1, 31, 38]). The reader familiar with these techniques may safely skip to Section 4.1. A *tree decomposition* of a graph G consists of a tree T and a set $X_v \subseteq V(G)$ for every $v \in V(T)$, such that:

1. $\bigcup_{v \in V(T)} X_v = V(G)$,
2. for every $xy \in E(G)$, there exists a node $v \in V(T)$ such that $\{x, y\} \subseteq X_v$, and
3. for every $x \in V(G)$, the nodes $\{v \in V(T) \mid x \in X_v\}$ induce a connected subgraph of T.

The *width* of a tree decomposition is the size of a largest set X_v minus one, and the minimum width over all tree decompositions is the *treewidth* of G. In an algorithmic context, a root node is chosen for the tree decomposition and dynamic programming is used to compute certain information bottom up from the leaf nodes to the root node, using appropriate dynamic programming rules. If the information is known for all child nodes of a node v, then a dynamic programming rule states how to use this to compute the desired information for v. Defining and proving dynamic programming rules for all possible cases that occur in general tree decompositions is rather complex, so usually the (rooted) tree decomposition is first transformed into a *nice tree decomposition*. This is a rooted tree decomposition that contains only four types of nodes:

- *leaf nodes* with no children,
- *join nodes* u with two children v and w such that $X_u = X_v = X_w$,
- *introduce nodes* u with one child v such that $X_v \subset X_u$ and $|X_u \setminus X_v| = 1$, and
- *forget nodes* u with one child v such that $X_u \subset X_v$ and $|X_v \setminus X_u| = 1$.

It is well-known that general tree decompositions can be transformed into nice tree decompositions in polynomial time (without increasing the tree-width) [31, 38]. This is a useful fact since then dynamic programming rules need only be defined for the four types of nodes above. In Section 6.1 we go into more detail about nice tree decompositions and slightly adjust their definition by interpreting leaf, forget, introduce and join nodes in the context of terminal graphs instead of graphs. (It is a good exercise for the reader to prove that both definitions are equivalent. However, since our presentation is self-contained, this step is not necessary for our proofs.)

4.1 Dynamic Programming Rules for Recoloring

We will now focus on CSGs for the k-color graph concept $\mathcal{C}_k$, using the terminal projection $p(G, T, \gamma) = \gamma|_T$. We will show how to compute the CSG $\mathcal{C}_k^c(G, T)$ when (G, T) is obtained using a forget, introduce or join operation from a (pair of) graph(s) for which we know the

CSG(s). Hatanaka, Ito and Zhou [20] considered a variant of these CSGs, namely for the case that $|T| = 1$ in the context of list colorings of caterpillars. They gave a combined introduce and forget rule and a restricted type of join rule, similar to the rules below.

We first state the (trivial) rule for computing $\mathcal{C}_k^c(G,T)$ for leaves, which follows from the facts that $\mathcal{C}_k(G)$ has k-colorings of G as nodes and that the label $\ell(x)$ of a node x in $\mathcal{C}_k^c(G,T)$ is a k-coloring of $G[T]$. Afterwards we give the rules for the forget, introduce and join operations, which can be proven using the properties of Lemma 1. Together these four rules allow us to compute the CSG $\mathcal{C}_k^c(G,T)$ for every terminal graph (G,T). Figure 2 illustrates the rules for the forget and introduce operation, whereas Figure 3 illustrates the rule for the join operation.

Lemma 3 (Leaf) *Let (G,T) be a terminal graph with $T = V(G)$. Then $\mathcal{C}_k^c(G,T)$ is isomorphic to $\mathcal{C}_k(G)$ and its label function ℓ is the isomorphism from $\mathcal{C}_k^c(G,T)$ to $\mathcal{C}_k(G)$. Moreover, for every k-coloring γ of G, the γ-node of $\mathcal{C}_k^c(G,T)$ is the node v with $\ell(v) = \gamma$.*

Lemma 4 (Forget) *Let (G,T) be a terminal graph. For every $v \in T$, it holds that $(H',\ell') = \mathcal{C}_k^c(G, T \setminus \{v\})$ can be computed from $(H,\ell) = \mathcal{C}_k^c(G,T)$ as follows:*

- *For every node x in H with $\ell(x) = \gamma$, let $\ell'(x) = \gamma|_{T\setminus\{v\}}$.*
- *Iteratively contract every edge between two nodes x and y with $\ell'(x) = \ell'(y)$ and assign label $\ell'(z) := \ell'(x)$ to the resulting node z.*

Moreover, for any coloring γ of G, the γ-node of $\mathcal{C}_k^c(G, T \setminus \{v\})$ is the node that results from contracting the set of nodes that includes the γ-node of $\mathcal{C}_k^c(G,T)$.

Proof. Let S denote the certificate for (H,ℓ), so for every node x of H, S_x denotes the set of k-colorings of G (or solutions), such that these sets satisfy the properties stated in Lemma 1. In addition, for every coloring γ for which a γ-node x has been marked in H, we may assume that $\gamma \in S_x$. We will prove the statement using Lemma 1 again, by giving a certificate S' for (H',ℓ'), and proving that the five properties hold for these.

The graph H' is obtained by iteratively contracting edges of H, so every node y of H' corresponds to a connected set of nodes of H, which we will denote by M_y. So $\{M_y \mid y \in V(H')\}$ is a partition of $V(H)$. For every node $y \in V(H')$, we define $S'_y = \cup_{x\in M_y} S_x$.

For every k-coloring γ of G such that the γ-node $x \in V(H)$ is marked, we define the γ-node of H' to be the node y with $x \in M_y$. Clearly, $\gamma \in S'_y$ then holds, so this is correct. It now remains to verify that the solution sets S'_x satisfy the five properties stated in Lemma 1.

1. $\{S_x \mid x \in V(H)\}$ is a partition of the nodes of $\mathcal{C}_k(G)$ (Lemma 1 Property 1), and $\{M_y \mid y \in V(H')\}$ is a partition of $V(H)$, so $\{S'_y \mid y \in V(H')\}$ is a partition of the nodes $\mathcal{C}_k(G)$.
2. Consider a node $y \in V(H')$, with label $\ell'(y)$, which is a k-coloring of $G[T \setminus \{v\}]$. Every node $x \in M_y$ has a label $\ell(x)$ with $\ell(x)|_{T\setminus\{v\}} = \ell'(y)$, and for every $\gamma \in S_x$, it holds that $\gamma|_T = \ell(x)$ (Lemma 1 Property 2), and thus $\gamma|_{T\setminus\{v\}} = \ell'(y)$. Therefore, for every $\gamma \in S'_y$, it holds that $\gamma|_{T\setminus\{v\}} = \ell'(y)$.
3. Consider two adjacent nodes x and y in H'. This implies that there exists an edge ab between the node sets M_x and M_y of H. By definition, all nodes $a \in M_x$ have $\ell'(a) = \ell'(x)$, and all nodes $b \in M_y$ have $\ell'(b) = \ell'(y)$. So if $\ell'(x) = \ell'(y)$, then the edge ab should also have been contracted when constructing H', a contradiction. Hence $\ell'(x) \neq \ell'(y)$.
4. Consider a node x of H'. The node set M_x is connected, so for any two nodes $y, z \in M_x$, the subgraph of H induced by M_x contains a path from y to x. Edges ab of this path correspond to solution sets S_a and S_b that contain adjacent solutions (Lemma 1 Property 5). In

addition, all such solution sets S_a are connected in $\mathcal{C}_k(G)$ (Lemma 1 Property 4). Combining these facts shows that the new solution sets S'_x are connected in $\mathcal{C}_k(G)$.

5. Let z and z' be two nodes of H'. By construction, z and z' are adjacent if and only if there exist nodes $x \in M_z$ and $x' \in M_{z'}$ that are adjacent in H. Two such nodes x and x' are adjacent in H if and only if there exist solutions $\alpha \in S_x$ and $\alpha' \in S_{x'}$ that are adjacent in $\mathcal{C}_k(G)$ (Lemma 1 Property 5). Using the definition of S_z and $S_{z'}$, we conclude that z and z' are adjacent if and only if there exist solutions $\alpha \in S_z$ and $\alpha' \in S_{z'}$ that are adjacent in $\mathcal{C}_k(G)$. □

Lemma 5 (Introduce) *Let (G, T) be a terminal graph obtained from a terminal graph $(G - v, T \setminus \{v\})$ by introducing v. Then $(H', \ell') = C^c_k(G, T)$ can be computed as follows from $(H, \ell) = \mathcal{C}^c_k(G - v, T \setminus \{v\})$:*

- *For every node x of H with label $\ell(x)$, and every color $c \in \{1, \ldots, k\}$: if the (unique) function $\delta : T \to \{1, \ldots, k\}$ with $\delta(v) = c$ and $\delta|_T = \ell(x)$ is a coloring of $G[T]$ then introduce a node x_c with label $\ell'(x_c) = \delta$.*
- *For every pair of distinct nodes x_c and y_d: add an edge between them if and only if (1) $x = y$ or (2) xy is an edge in H and $c = d$.*

Moreover, for every k-coloring γ of G, if x is the $\gamma|_{V(G)\setminus\{v\}}$-node in H and $\gamma(v) = c$, then x_c is the γ-node of H'.

Proof. Let S be a certificate for (H, ℓ), so for every node x of H, let S_x denote the set of k-colorings of $G - v$ (or solutions), such that these sets satisfy the properties stated in Lemma 1. In addition, for every coloring γ for which a γ-node x has been marked in H, we may assume that $\gamma \in S_x$. Now we construct a certificate S' for (H', ℓ'). For every node x_c of H' (that corresponds to a node x of H, and to assigning a color c to the new vertex v), we define S'_{x_c} to be the set of k-colorings α of G with $\alpha(v) = c$ and $\alpha|_{T\setminus\{v\}} \in S_x$. For every k-coloring γ of G and node x_c of H', we define x_c to be the γ-node of H' if and only if $\gamma(v) = c$ and x is the $\gamma|_{V(G)\setminus\{v\}}$-node of H. Clearly, this guarantees $\gamma \in S'_{x_c}$ for the chosen γ-node x_c. To prove the statement, it only remains to show that the new solution sets S'_{x_c} satisfy the five properties stated in Lemma 1.

1. First, we observe that for every node x_c of H', S'_{x_c} is a nonempty set of k-colorings of G, because S_x is nonempty (Lemma 1), and by choice of c, every coloring $\alpha \in S_x$ can be extended to a k-coloring of G by setting $\alpha(v) = c$ (this uses the fact that $N(v) \subseteq T$). So to prove that the new solution sets form a partition of the nodes of $\mathcal{C}_k(G)$, it only remains to show that every k-coloring α of G is included in S'_{x_c} for exactly one new node x_c. For every such α, there exists a unique node x of H such that $\alpha|_{V(G)\setminus\{v\}} \in S_x$ (Lemma 1 Property 1). Since α is a coloring of G, $\alpha|_T$ is a coloring of $G[T]$, so we have created one node x_c with $c = \alpha(v)$. This is the unique node of H' with $\alpha \in S'_{x_c}$.
2. Consider a node x_c of H', with label $\ell'(x_c) = \delta$. For every $\alpha \in S'_{x_c}$, it holds that $\alpha(v) = c$ and $\delta(v) = c$. Furthermore, $\delta|_{T\setminus\{v\}} = \ell(x) = \alpha|_{T\setminus\{v\}}$ (Lemma 1 Property 2). This shows that the label $\ell'(x_c)$ is chosen correctly.
3. Consider two adjacent nodes x_c and y_d of H'. If $x = y$ then $c \neq d$, so $\ell'(x) \neq \ell'(y)$. Otherwise, x and y are adjacent nodes in H, so $\ell(x) \neq \ell(y)$ (Lemma 1 Property 3). The labels $\ell(x)$ and $\ell(y)$ are the restrictions of $\ell'(x_c)$ and $\ell'(y_d)$ to $T \setminus \{v\}$, so also in this case we conclude that $\ell'(x) \neq \ell'(y)$.

4. Consider a node x_c of H', and two k-colorings α and β in S'_{x_c}. There is a path P from $\alpha|_{V(G)\setminus\{v\}}$ to $\beta|_{V(G)\setminus\{v\}}$ in the subgraph of $\mathcal{C}_k(G-v)$ induced by S_x (Lemma 1 Property 4). As $N(v) \subseteq T$, all colorings γ in P have an *extension* $\gamma' \in S_{x_c}$ with $\gamma'(v) = c$ and $\gamma'|_{V(G)\setminus\{v\}} = \gamma$. So replacing all colorings in P by their extension this way yields a path from α to β in the subgraph of $\mathcal{C}_k(G)$ induced by S'_{x_c}. Therefore, S'_{x_c} is connected.
5. Consider two distinct nodes x_c and y_d in H', and their corresponding sets of solutions S'_{x_c} and S'_{y_d}. Observe that these contain solutions that are adjacent in $\mathcal{C}_k(G)$ if and only if at least one of the following is true: (1) $c = d$ (and thus $x \neq y$) and S_x and S_y contain solutions that are adjacent in $\mathcal{C}_k(G-v)$, or (2) $c \neq d$ and $S_x \cap S_y \neq \emptyset$. The first case holds if and only if $c = d$ and the nodes x and y are adjacent in H (Lemma 1 Property 5). In the second case, $S_x \cap S_y \neq \emptyset$ holds if and only if $S_x = S_y$, and thus $x = y$ (Lemma 1 Property 1). This shows that we have added the edges correctly. □

Lemma 6 (Join) *Let (G,T) be a terminal graph that is the join of terminal graphs (G_1,T) and (G_2,T). Let $(H_1,\ell_1) = C^c_k(G_1,T)$ and $(H_2,\ell_2) = C^c_k(G_2,T)$. Then $(H,\ell) = C^c_k(G,T)$ can be computed as follows:*

- *For every pair of nodes $x \in V(H_1)$ and $y \in V(H_2)$: if $\ell_1(x) = \ell_2(y)$ then introduce a node (x,y) with $\ell((x,y)) = \ell_1(x)$.*
- *For two distinct nodes (x,y) and (x',y'), add an edge between them if and only if xx' is an edge in H_1 and yy' is an edge in H_2.*

Moreover, for every k-coloring γ of G, if x is the $\gamma|_{V(G_1)}$-node in H_1 and y is the $\gamma|_{V(G_2)}$-node in H_2, then (x,y) is the γ-node in H.

Proof. Denote $V_1 = V(G_1)$ and $V_2 = V(G_2)$. For every node x of H_1, let S^1_x denote the set of k-colorings of G_1 such that these sets satisfy the properties stated in Lemma 1. Similarly, we define the sets S^2_x for every node x of H_2. In addition, we assume again that these sets coincide with the choices of $\gamma|_{V_1}$-nodes and $\gamma|_{V_2}$-nodes.

We define a certificate S for (H,ℓ) as follows. For every node (x,y) of H, we define the set $S_{(x,y)}$ to consist of all k-color assignments α of G such that $\alpha|_{V_1} \in S^1_x$ and $\alpha|_{V_2} \in S^2_y$. For any k-coloring γ of G and node (x,y) of H, we choose (x,y) to be the γ-node of H if and only if x is the $\gamma|_{V_1}$-node of H_1 and y is the $\gamma|_{V_2}$-node of H_2. This obviously guarantees that $\gamma \in S_{(x,y)}$ for the chosen γ-node (x,y). To prove the statement, it only remains to show that the new solution sets $S_{(x,y)}$ satisfy the five properties stated in Lemma 1.

1. First, we show that for every node (x,y) of H, $S_{(x,y)}$ is a nonempty set of k-colorings of G. The set S^1_x contains at least one coloring α_1 of G_1, and S^2_y contains at least one coloring α_2 of G_2 (Lemma 1 Property 1). Both of these colorings yield the coloring $\ell((x,y)) = \ell_1(x) = \ell_2(y)$ when restricted to T (Lemma 1 Property 2), so they can be combined into a k-color assignment α for G. Since all edges of G are part of G_1 or G_2 (by definition of the join operation), the resulting α is a k-coloring of G.
   To prove that the sets $S_{(x,y)}$ partition the k-colorings of G, it now suffices to show that every k-coloring α of G is included in exactly one set $S_{(x,y)}$. Consider $\alpha_i = \alpha|_{V_i}$ for $i = 1,2$. Then $\alpha_1 \in S_x$ for exactly one node x of H_1, and $\alpha_2 \in S_y$ for exactly one node y of H_2 (Lemma 1 Property 1). These nodes have $\ell_1(x) = \alpha|_T$ and $\ell_2(y) = \alpha|_T$ (Lemma 1 Property 2), so we have created exactly one node (x,y) with $\alpha \in S_{(x,y)}$.
2. Consider a node (x,y) of H, and a solution $\alpha \in S_{(x,y)}$. Let $\alpha_1 = \alpha|_{V_1}$. Then $\alpha|_T = \alpha_1|_T = \ell_1(x) = \ell((x,y))$ (Lemma 1 Property 2).

3. Consider adjacent nodes (x, y) and (x', y') of H. Then by definition, x and x' are adjacent in H_1, so $\ell_1(x) \neq \ell_2(x')$ (Lemma 1 Property 3), and thus $\ell((x, y)) \neq \ell((x', y'))$.
4. Consider a node (x, y) of H. We prove that $S_{(x,y)}$ is a connected set in $\mathcal{C}_k(G)$. Consider any two colorings $\alpha, \beta \in S_{(x,y)}$. Define $\alpha_i = \alpha|_{V_i}$ and $\beta_i = \beta|_{V_i}$ for $i = 1, 2$. Then for $i = 1, 2$, there exists a path P^i (or *recoloring sequence*) from α_i to β_i, in the subgraph of $\mathcal{C}_k(G_i)$ induced by S^1_x resp. S^2_y (Lemma 1 Property 4). All colorings γ in both paths satisfy $\gamma|_T = \ell((x, y) = \ell_1(x) = \ell_2(y)$ (Lemma 1Property 2). Therefore, we can construct a recoloring sequence from α to β that contains only colorings in $S_{(x,y)}$ by first recoloring vertices of $V_1 \setminus T$ as prescribed by the recoloring sequence P^1 (which yields a coloring δ of G with $\delta|_{V_1} = \beta_1$ and $\delta|_{V_2} = \alpha_2$), and subsequently recoloring vertices of $V_2 \setminus T$ as prescribed by the recoloring sequence P^2 (which yields the coloring β). This can be done because $V_1 \cap V_2 = T$ and neither P^1 nor P^2 recolors a vertex of T. All of the color assignments in the resulting sequence are part of $S_{(x,y)}$ by definition (and they are in fact colorings, as argued above in (1)).
5. Consider two distinct nodes (x, y) and (x', y') in H. We prove that they are adjacent if and only if there exist solutions $\alpha \in S_{(x,y)}$ and $\beta \in S_{(x',y')}$ that are adjacent in $\mathcal{C}_k(G)$.

   Suppose (x, y) and (x', y') are adjacent. By definition, this means that x and x' are adjacent (and thus distinct) nodes of H_1, and y and y' are adjacent nodes of H_2. So we can choose solutions $\alpha^1 \in S^1_x$ and $\beta^1 \in S^1_{x'}$ that are adjacent in $\mathcal{C}_k(G_1)$, and solutions $\alpha^2 \in S^2_y$ and $\beta^2 \in S^2_{y'}$ that are adjacent in $\mathcal{C}_k(G_2)$ (Lemma 1 Property 5). Since $\ell_1(x) \neq \ell_1(x')$ (Lemma 1 Property 3), and $\alpha^1|_T = \ell_1(x)$ and $\beta^1|_T = \ell_2(x')$ (Lemma 1 Property 2), the colorings α^1 and β^1 differ on T, and therefore, since they are adjacent, only on T (so their restrictions to $V_1 \setminus T$ are the same). Similarly, the colorings α^2 and β^2 differ only on T. By definition of (x, y), $\ell_1(x) = \ell_2(y)$, so we can choose a k-coloring α of G with $\alpha|_{V_1} = \alpha^1$ and $\alpha|_{V_2} = \alpha^2$. Similarly, we can choose a k-coloring β of G with $\beta|_{V_1} = \beta^1$ and $\beta|_{V_2} = \beta^2$. As argued above, the colorings α and β differ only on one vertex in T, so they are adjacent in $\mathcal{C}_k(G)$. By their construction, $\alpha \in S_{(x,y)}$ and $\beta \in S_{(x',y')}$, so this proves the first direction.

   For the converse, suppose that there exist adjacent colorings $\alpha \in S_{(x,y)}$ and $\beta \in S_{(x',y')}$. Let $\alpha^i = \alpha|_{V_i}$ and $\beta^i = \beta|_{V_i}$ for $i = 1, 2$. Let w be the (unique) vertex of G with $\alpha(w) \neq \beta(w)$. If $w \in T$ then, by using similar arguments as in the previous paragraph, one can verify that x and x' are adjacent nodes in H_1 and y and y' are adjacent nodes in H_2, and therefore (x, y) and (x', y') are adjacent in H. We conclude the proof by showing that $w \in T$ always holds. Suppose for contradiction that $w \notin T$; without loss of generality assume that $w \in V_1 \setminus T$. So $\alpha^2 = \beta^2$. Therefore $S^2_y \cap S^2_{y'} \neq \emptyset$, and thus $y = y'$ (Lemma 1 Property 1). It follows that $\ell_1(x) = \ell_2(y) = \ell_2(y') = \ell_1(x')$. In addition, since (x, y) and (x', y') are distinct nodes and $y = y'$, it follows that $x \neq x'$. But $\alpha^1 \in S^1_x$ and $\beta^1 \in S^2_{x'}$ are adjacent, so $xx' \in E(H_1)$ (Lemma 1 Property 5). This is a contradiction to the fact that $\ell_1(x) \neq \ell_1(x')$ must hold due to Lemma 1 Property 3. □

Remark 1. The DP rules in this section can be generalized further to capture the rules of [20] for the *list coloring* generalization $\mathcal{C}_L$ of $\mathcal{C}_k$. In this generalization, an instance G, L consists of a graph G together with color lists $L(v) \subseteq \{1, \ldots, k\}$ for each $v \in V(G)$. Solutions are now list colorings, which are colorings α of G such that $\alpha(v) \in L(v)$ for each $v \in V(G)$. Adjacency is defined as before. So the *list coloring solution graph* $\mathcal{C}_L(G, L)$ is an induced subgraph of $\mathcal{C}_k(G)$. Hence, it is straightforward to generalize our DP rules to $\mathcal{C}_L$, namely by simply omitting all nodes that correspond to invalid vertex colors.

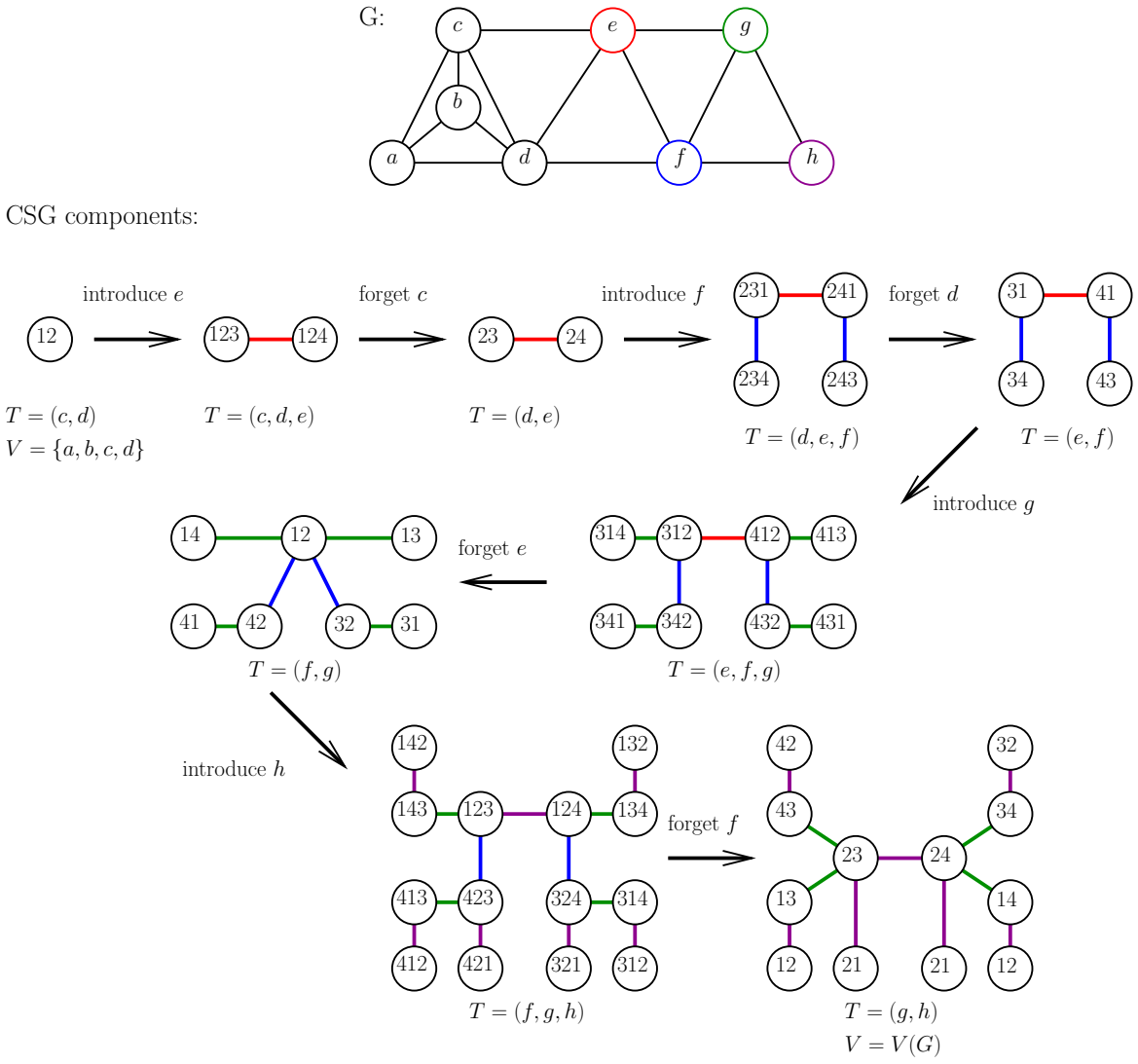


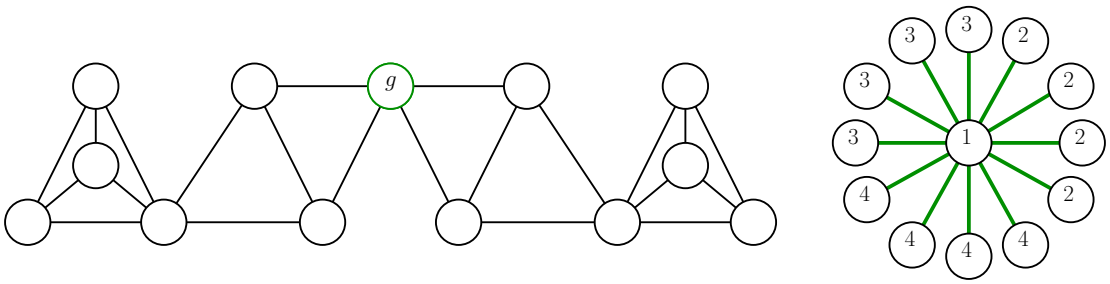
Fig. 2. An example of computing CSGs using forget and introduce operations. A 4-colorable 2-connected chordal graph G with $V(G) = \{a, b, c, d, e, f, g, h\}$ is shown. Note that G is in fact unit interval and isomorphic to the graph G_8^I defined in Section 4. Starting with one component of the CSG $\mathcal{C}_4^c(G[\{a, b, c, d\}], \{c, d\})$, the corresponding component of $\mathcal{C}_4^c(G, \{g, h\})$ is computed, using four forget and introduce operations. The $G[T]$-colorings in the node labels are given as sequences of colors for the ordered version of T as indicated below each CSG. For instance, for $T = (c, d)$, the node label 12 indicates the coloring γ with $\gamma(c) = 1$ and $\gamma(d) = 2$.

5 Examples of Exponential Size CSGs

In this section we further illustrate the dynamic programming rules given in Section 4 and show that components of $\mathcal{C}_k^c(G)$ can grow exponentially, even if G is a chordal graph and $k = 4$. Namely, when considering 4-colorable chordal graphs that may have cut vertices, it is easy to obtain CSGs that have exponentially large components: take p copies of the graph shown in Figure 1(a), and identify the g-vertices of all of these graphs. Call the resulting graph G_p^*. The graph G_2^* is shown in Figure 3(a).

Proposition 7 *For every integer $p \geq 1$, $\mathcal{C}_4^c(G_p^*, \{g\})$ has a component with $1 + 3 \cdot 2^p$ nodes.*

Proof. By induction over p we prove the following: $\mathcal{C}_4^c(G_p^*, \{g\})$ has a component that is a star (a $K_{1,n}$ graph) in which the central node has label 1 (to be precise, this means that the label is a coloring that assigns color 1 to vertex g), and which has 2^p leaves with label j, for $j \in \{2, 3, 4\}$. The case $p = 1$ can easily be verified; see also Figure 1. For the induction step, apply Lemma 6 to the star components of $\mathcal{C}_4^c(G_{p-1}^*, \{g\})$ and $\mathcal{C}_4^c(G_1^*, \{g\})$ given by the induction hypothesis: for $j \in \{2, 3, 4\}$, the 2^{p-1} nodes with label j of the former graph are combined with two nodes with label j of the latter graph, giving 2^p new nodes with label j. All of these are adjacent only to the unique new node with label 1. □



(a) Graph G_2^* (b) $\mathcal{C}_4^c(G_2^*, \{g\})$

Fig. 3. (a) The graph G_2^*, which can be obtained using a join operation on two subgraphs isomorphic to the graph G shown in Figure 1(a), with $T = \{g\}$. (b) One component of the CSG $\mathcal{C}_4^c(G_2^*, \{g\})$, which can be obtained by combining two copies of the CSG from Figure 1(d), as shown in Lemma 6.

With a little more effort, we can also construct CSGs with exponentially large components when restricting to $(k-2)$-connected k-colorable chordal graphs, or even 2-connected 4-colorable unit interval graphs, as we show below.

A graph G is an *interval graph* if G has a representation in which each vertex corresponds to an interval of the line, and two vertices are adjacent if and only if their corresponding intervals intersect. If these intervals have unit length, then G is a *unit interval graph*. For $p \geq 4$, let the graph G_p^I have vertex set $\{v_0, \ldots, v_{p-1}\}$ and edge set $\{v_0v_3\} \cup \{v_iv_{i+1} \mid 0 \leq i \leq p-2\} \cup \{v_iv_{i+2} \mid 0 \leq i \leq p-3\}$. A graph isomorphic to G_8^I is shown in Figure 2. It is readily seen that each G_p^I is an interval graph. An interval graph is a unit interval if and only if it is *claw-free*, that is, does not contain the 4-vertex star as an induced subgraph [40]. Hence, as each G_p^I is claw-free, each G_p^I is even a unit interval graph.

For our proof we need the following simple observation (which has been used in various earlier papers on recoloring, such as in the proof of Theorem 11 of [8]).

Lemma 8 *Let α and β be two k-colorings of a graph $G = (V, E)$, and let v be a vertex of degree at most $k-2$. Then $\mathcal{C}_k(G)$ contains a path from α to β if and only if $\mathcal{C}_k(G-v)$ contains a path from $\alpha|_{V\setminus\{v\}}$ to $\beta|_{V\setminus\{v\}}$.*

Below we state our claim more precisely and give a proof of it as well.

Proposition 9 *For $p = 4q + 4$ with $q \in \mathbb{N}$, the CSG $\mathcal{C}_4^c(G_p^I, \{v_{p-2}, v_{p-1}\})$ has 4! components on at least 2^q nodes.*

Proof. For every set $S \subseteq \{1, \ldots, q\}$, we construct a coloring α_S of G_p^I as follows. For all $j \in \{0, \ldots, q\}$:

- $\alpha_S(v_{4j}) = 3$ if $j \in S$, and $\alpha_S(v_{4j}) = 4$ if $j \notin S$.
- $\alpha_S(v_{4j+1}) = 4$ if $j \in S$, and $\alpha_S(v_{4j+1}) = 3$ if $j \notin S$.
- $\alpha_S(v_{4j+2}) = 1$.
- $\alpha_S(v_{4j+3}) = 2$.

Observe that for every S, α_S is a 4-coloring of G_p^I. There are 2^q possible choices of S, and therefore 2^q such colorings α_S. An induction proof based on Lemma 8 shows that for every $S_1 \subseteq \{1, \ldots, q\}$ and $S_2 \subseteq \{1, \ldots, q\}$, $\mathcal{C}_4(G)$ contains a path from α_{S_1} to α_{S_2}. Informally, vertex v_{p-1} has degree $2 = 4 - 2$ and we may delete v_{p-1} by Lemma 8. After deleting v_{p-1}, v_{p-2} has degree 2, and may be deleted next. Continuing this procedure ends with two 4-colorings of the complete graph on vertices $\{v_0, v_1, v_2, v_3\}$, which coincide. That is, for every S, α_S assigns the colors $4, 3, 1, 2$ to the vertices v_0, v_1, v_2, v_3, respectively (note that no vertex of $\{v_0, v_1, v_2, v_3\}$ can be recolored). It follows that all of the colorings α_S we constructed are part of the same component of $\mathcal{C}_4(G_p^I)$. Finally, we observe that every coloring α_S forms a one-node label component in $\mathcal{C}_4^c(G_p^I, \{v_{p-2}, v_{p-1}\})$. Indeed, the only vertex that can be recolored in any α_S is the vertex v_{p-1}; all other vertices have three distinctly-colored neighbors. Summarizing, $\mathcal{C}_4^c(G_p^I, \{v_{p-2}, v_{p-1}\})$ contains a component that contains at least 2^q nodes that are labeled with a coloring that assigns colors 1 and 2 to vertices v_{p-2} and v_{p-1}, respectively.

For every 4-coloring of $G[\{v_0, v_1, v_2, v_3\}]$, the CSG contains a component isomorphic to the component considered above, so there are 4! components of this type. □

The last CSG shown in Figure 2 contains two nodes with label 12; these correspond to the colorings $\alpha_\emptyset$ and $\alpha_{\{1\}}$ constructed in the above proof. The CSG shows that any recoloring sequence between these two colorings needs to recolor the vertices $v_{p-2} = g$ and $v_{p-1} = h$ at least two resp. three times. We remark that the proofs of Propositions 7 and 9 illustrate different proof techniques for CSGs: one uses the dynamic programming rules, and the other argues about label components of the solution graph directly. Both examples show that we need to do more than only computing CSGs to solve the problem for $(k-2)$-connected chordal graphs in polynomial time. In the next section we will characterize the CSGs and show that it suffices to compute only a part of them.

6 Recoloring Chordal Graphs

In this section we will show that CSGs can be used to efficiently decide the $\mathcal{C}_k$-Reachability problem for $(k-2)$-connected chordal graphs. Recall that a graph is chordal if it contains no

induced cycle of length greater than 3. To prove the result, we use the fact that for a chordal graph G and any clique T of G, the terminal graph (G, T) can recursively be constructed from simple cliques using a polynomial number of clique-based introduce, forget and join operations. In order to do this we first define the notion of a nice tree decomposition for terminal graphs in Section 6.1. In the same section we show an upper bound on the size of an arbitrary nice tree decomposition for a terminal graph. Afterwards we make the above fact for chordal graphs precise in Section 6.2, namely by defining chordal nice tree decompositions for terminal graphs. We remark that some of our statements are similar to (and can alternatively be deduced from) well-known facts about tree decompositions [16] and nice tree decompositions [31] for graphs. However, for readability, and since we need to prove an upper bound on the size of a nice tree decomposition for terminal graphs, we give a self-contained presentation.

6.1 Nice Tree Decompositions for Terminal Graphs

Nice tree decompositions describe how a terminal graph (G, T) can be obtained from trivial graphs using forget, introduce and join operations. A *nice tree decomposition* of a terminal graph (G, T) (where G is an arbitrary graph, not necessarily chordal, and T may not be a clique) is a tuple $(\mathcal{T}, X, r)$, where $\mathcal{T}$ is a tree with root r and X is an assignment of *bags* $X_u \subseteq V(G)$ for each $u \in V(\mathcal{T})$ that can be defined recursively as follows:

(1) If $T = V(G)$, then the tree $\mathcal{T}$ consists of one (root) node r with bag $X_r = T$.
(2) If $v \in V(G) \setminus T$ and $(\mathcal{T}', X, r')$ is a nice tree decomposition of $(G, T \cup \{v\})$, then a nice tree decomposition for (G, T) can be obtained by adding a new root r with $X_r = T$ and adding the edge rr'.
(3) If (G, T) can be obtained from $(G - v, T \setminus \{v\})$ using an introduce operation and $(\mathcal{T}', X, r')$ is a nice tree decomposition of $(G - v, T \setminus \{v\})$, then a nice tree decomposition for (G, T) can be obtained by adding a new root r with $X_r = T$ and adding the edge rr'.
(4) If (G, T) can be obtained from (G_1, T) and (G_2, T) using a join operation, and $(\mathcal{T}_1, X, r_1)$ and $(\mathcal{T}_2, X, r_2)$ are nice tree decompositions of (G_1, T) and (G_2, T), then a nice tree decomposition for (G, T) can be obtained by adding a new root r with $X_r = T$ and adding edges rr_1 and rr_2.

We call a node $u \in V(\mathcal{T})$ a *leaf*, *forget node*, *introduce node* or *join node* if u is added as the root in case (1), (2), (3) or (4), respectively. The *width* of $(\mathcal{T}, X, r)$ is $\max_{u \in V(\mathcal{T})} |X_u| - 1$.

We will need the following lemma.

Lemma 10 *Let $(\mathcal{T}, X, r)$ be a nice tree decomposition of a terminal graph (G, T) of width at most $w \geq 1$, and let $n = |V(G)| \geq 1$. Then $|V(\mathcal{T})| \leq (w + 4)n$.*

Proof. Let $t = |T|$. We use induction over $|V(\mathcal{T})|$ to prove that

$$|V(\mathcal{T})| \leq 2n - t + (w + 2)(\max\{0, n - t - 1\}).$$

Then, since this value is at most $(w + 4)n$, the lemma statement follows.

Let $|V(\mathcal{T})| = 1$. Then the root r of $\mathcal{T}$ is a leaf (so $T = V(G)$ and $t = n$). This means that

$$|V(\mathcal{T})| = 1 \leq n = 2n - t + (w + 2)(\max\{0, n - t - 1\}).$$

Let $|V(\mathcal{T})| \geq 2$. Then the root r is either a forget, introduce or join node. We consider each of these cases below.

If r is a forget node then by induction, after adding the new root to the tree, the number of nodes is at most:

$$1 + 2n - (t+1) + (w+2)(\max\{0, n-(t+1)-1\}) \;\leq\; 2n - t + (w+2)(\max\{0, n-t-1\}).$$

If r is an introduce node then by induction, after adding the new root to the tree, the number of nodes is at most:

$$1 + 2(n-1) - (t-1) + (w+2)(\max\{0, n-1-(t-1)-1\}) =$$

$$2n - t + (w+2)(\max\{0, n-t-1\}).$$

Finally, suppose that r is a join node and that (G,T) is obtained by joining together graphs on n_1 and $n_2 = n - n_1 + t$ nodes. From the definition of the join operation it follows that both of these values are strictly larger than t, so we may write $\max\{0, n_1 - t - 1\} = n_1 - t - 1$ and $\max\{0, n_2 - t - 1\} = n_2 - t - 1 = n - n_1 - 1$. Then by induction, after adding the new root, the number of nodes is at most:

$$1 \;+\; 2n_1 - t + (w+2)(n_1 - t - 1) \;+\; 2(n - n_1 + t) - t + (w+2)(n - n_1 - 1) \;=$$

$$1 + 2n + (w+2)(n-t-2) \;\leq\; 2n - t + (w+2)(n-t-1).$$

For the last step, we used the fact that $t \leq (w+1)$. □

6.2 Chordal Nice Tree Decompositions for Terminal Graphs

A nice tree decomposition $(\mathcal{T}, X, r)$ of (G,T) is *chordal* if for every node $u \in V(\mathcal{T})$, X_u is a clique of G. Note that, if $(\mathcal{T}, X, r)$ is a chordal nice tree decomposition of a k-colorable graph G, then the width of $(\mathcal{T}, X, r)$ is at most $k-1$. Hence, Lemma 10 shows that any chordal nice tree decomposition of a k-colorable graph has at most $(k+3)n$ nodes.

In order to show how to find a chordal nice tree decomposition in polynomial time we need the following lemma, which tells us how to select the proper type of root node when constructing such a tree decomposition. Here, a terminal graph (G_1, T_1) is called *smaller* than another terminal graph (G_2, T_2) if $2|V(G_1)| - |T_1| < 2|V(G_2)| - |T_2|$.

Lemma 11 *Let (G,T) be a terminal graph where G is a chordal graph, and T is a clique with $T \neq V(G)$. If $G - T$ is disconnected, then (G,T) can be obtained from a pair of smaller chordal terminal graphs (G_1, T) and (G_2, T) using a join operation. Otherwise, (G,T) can be obtained from a smaller chordal terminal graph (G', T') using either a forget or introduce operation, where T' is a clique. For every such (G,T), the relevant operation and subgraph(s) can be found in polynomial time.*

Proof. If $G - T$ is disconnected, then let C be the vertex set of a component of $G - T$, and consider the two graphs $G_1 = G[T \cup C]$ and $G_2 = G - C$. Then (G,T) is the join of (G_1, T) and (G_2, T) with $|V(G_1)| - |T| < |V(G)| - |T|$ and $|V(G_2)| - |T| < |V(G)| - |T|$.

Next assume that $G - T$ is connected. If T contains a vertex v that has no neighbors in $G - T$, then (G,T) can be obtained from $(G - v, T \setminus \{v\})$ using an introduce operation (note that this operation requires that v has no neighbors in $G - T$), and $2|V(G - v)| - |T \setminus \{v\}| = 2|V(G)| - |T| - 1$.

Finally, assume that $G - T$ is connected and every vertex in T is adjacent to at least one vertex in $G - T$. Below we prove that there exists a vertex $u \in V(G) \setminus T$ that is adjacent to every vertex of T. As T is a clique, this means that $T \cup \{u\}$ is a clique. Consequently, (G, T) can be obtained from $(G, T \cup \{u\})$ by a forget operation, such that $T \cup \{u\}$ is a clique in G, and $2|V(G)| - |T \cup \{u\}| = 2|V(G)| - |T| - 1$.

We choose $u \in V(G) \setminus T$ to be a vertex with a maximum number of neighbors in T over all vertices in $G - T$. We claim that u is adjacent to every vertex of T. For contradiction, assume there exists a vertex $y \in T$ that is not adjacent to u. Recall that $G - T$ is connected and that every vertex in T is adjacent to at least one vertex in $G - T$. These two assumptions imply that we can choose a *shortest* path P in $G - T$ from u to a neighbor v of y (so v is the only vertex of P that is adjacent to y).

Let u' be the last vertex on P (when going from u to v) that has a neighbor $z \in T$ not adjacent to v. As u is a vertex in $G - T$ with the maximum number of neighbors in T over all vertices in $G - T$ and u is not adjacent to $y \in T \cap N(v)$, vertex u satisfies this condition. Hence, vertex u' exists (and $u' \neq v$).

Let P' be the subpath of P from u' to v. By our choice of u', no vertex of P' other than u' is adjacent to z. Recall also that v is the only vertex of P' that is adjacent to y. As T is a clique, y and z are adjacent. As P' is a shortest path, the subgraph of G induced by $V(P')$ is a path. As $u' \neq v$, it follows that $|V(P')| \geq 2$. Hence, $V(P') \cup \{y, z\}$ induces a cycle of length at least 4 in G, contradicting the fact that G is chordal. We conclude that u is adjacent to every vertex of T, as desired.

The above case study can easily be translated to a polynomial-time algorithm for finding the graph operation that applies. □

We are now ready to state the following result.

Corollary 12 *Let G be a chordal k-colorable graph on n vertices, and let T be a clique of G. Then it is possible to find a chordal nice tree decomposition of (G, T) on at most $(k+3)n$ nodes in polynomial time.*

Proof. Lemma 11 shows how we can choose the proper type of root node. We can build the chordal nice tree decomposition by adding this node to the tree decomposition(s) of (a) smaller graph(s). The entire chordal nice tree decomposition is constructed by continuing this process recursively. Lemma 10 shows that the resulting chordal nice tree decomposition has at most $(w + 4)n$ nodes, where $w + 1$ is the maximum bag size. Since every bag is a clique of G and the graph is k-colorable, we have $w + 1 \leq k$, so there are at most $(k + 3)n$ nodes. Since we have a polynomial number of nodes, and for every node we spend polynomial time (Lemma 11), the entire process terminates in polynomial time. □

We note that the precise complexity bound in Corollary 12 depends on implementation details, which are beyond the scope of this paper.

6.3 The Structure of CSGs for $(k-2)$-Connected Chordal Graphs

Using an inductive proof based on Lemma 11, we will now characterize the shape of CSGs for $(k-2)$-connected k-colorable chordal graphs. We start with a lemma that we will apply to $(k-2)$-connected k-colorable chordal graphs in our induction proofs.

Lemma 13 *Let G be a ℓ-connected chordal graph, and let T be a clique of G with $T \neq V(G)$. If (G, T) can be obtained from $(G - v, T \setminus \{v\})$ using an introduce operation, then $|T| \geq \ell + 1$ and $G - v$ is ℓ-connected. If (G, T) can be obtained from (G_1, T) and (G_2, T) using a join operation, then $|T| \geq \ell$ and both G_1 and G_2 are ℓ-connected.*

Proof. If (G, T) is obtained from $(G - v, T \setminus \{v\})$ using an introduce operation, then $N(v) \subseteq T \setminus \{v\}$ by definition. Since $T \neq V(G)$, it follows that $T \setminus \{v\}$ is a vertex cut set of G that separates v from at least one other vertex, so $|T| = |T \setminus \{v\}| + 1 \geq \ell + 1$. In addition, since T is a clique of G, every vertex cut set of $G - v$ is also a vertex cut set of G, and therefore $G - v$ is ℓ-connected.

If (G, T) is obtained from (G_1, T) and (G_2, T) using a join operation, then T is a vertex cut set of G that separates $V(G_1) \setminus T$ from $V(G_2) \setminus T$, so $|T| \geq \ell$. In addition, since T is a clique of G, every vertex cut set of G_i is a vertex cut set of G (for $i = 1, 2$), and therefore G_1 and G_2 are ℓ-connected. □

We need some extra definitions. For integers m, k with $1 \leq m \leq k$, a labeled graph (H, ℓ) is an (m, k)-*color-complete graph* if there exists a complete graph $K[T]$ on vertex set T with $|T| = m$ such that:

- for all vertices $v \in V(H)$, $\ell(v)$ is a k-coloring of $K[T]$,
- every such k-coloring of $K[T]$ appears at exactly one vertex of H, and
- two vertices of H are adjacent if and only if their labels differ on exactly one vertex of T.

From this definition it follows that for every pair of integers m and k, there is a unique (m, k)-color complete graph, up to the choice of T. An (m, k)-color-complete graph has $k!/(k - m)!$ vertices (this is the number of ways to k-color a complete graph on m vertices), and every vertex has degree $m(k - m)$. In particular, if $m = k$ then the graph consists of $k!$ isolated vertices (meaning that the graph is a forest). A labeled graph (H, ℓ) is said to satisfy the *injective neighborhood property* (INP) if for every vertex $u \in V(H)$ and every pair of distinct neighbors $v, w \in N(u)$, it holds that $\ell(v) \neq \ell(w)$. Note that (m, k)-color-complete graphs trivially satisfy the INP.

We will now show that for the graphs we consider, the following invariant is maintained by introduce, forget and join operations:

The CSG is an (m, k)-color complete graph, or a forest that satisfies the INP.

Note that a (k, k)-color complete graph is trivially a forest that satisfies the INP. We start with the trivial observation that this invariant initially holds.

Lemma 14 *Let $G = (V, E)$ be a complete graph on m vertices, with $m \leq k$. Then $\mathcal{C}_k^c(G, V)$ is an (m, k)-color complete graph.*

We now prove that a *forget* operation maintains the invariant (below, we argue that all the relevant cases are covered by the next lemma). Recall that a label $\ell(u)$ of a node u of $\mathcal{C}_k^c(G, T)$ is a coloring of $G[T]$, so by $\ell(u)(x)$ we denote the color that $x \in T$ receives in this coloring.

Lemma 15 *Let G be a k-colorable chordal graph and let T be a clique of G with $k - 1 \leq |T|$, and $v \in T$. If $\mathcal{C}_k^c(G, T)$ is a $(k-1, k)$-color complete graph, then $\mathcal{C}_k^c(G, T \setminus \{v\})$ is a $(k-2, k)$-color complete graph. If $\mathcal{C}_k^c(G, T)$ is a forest that satisfies the INP, then $\mathcal{C}_k^c(G, T \setminus \{v\})$ is a forest that satisfies the INP.*

Proof. Let $(H, \ell) = \mathcal{C}_k^c(G, T)$ and $(H', \ell') = \mathcal{C}_k^c(G, T \setminus \{v\})$. We will use the fact that (H', ℓ') can be constructed from (H, ℓ) as shown in Lemma 4.

First consider the case that (H, ℓ) is a $(k-1, k)$-color-complete graph. Then, from the definition of $(k-1, k)$-color-complete graphs it follows that for every coloring α of $G[T \setminus \{v\}]$, the nodes $\{x \in V(H) \mid \ell(x)|_{T \setminus \{v\}} = \alpha\}$ induce a nonempty complete subgraph of H. When constructing (H', ℓ') from (H, ℓ), this subgraph will be contracted into one node, so for every such coloring α, H' contains exactly one node with label α. Consider two colorings α_1 and α_2 of $G[T \setminus \{v\}]$ that differ on only one vertex $w \in T \setminus \{v\}$. We can extend both to a coloring of $G[T]$ by choosing a color for v that occurs in neither α_1 nor α_2 (since $|T \setminus \{v\}| = k-2$), which yields colorings of $G[T]$ that are adjacent in H (since (H, ℓ) is $(k-1, k)$-color complete) and that are compatible with α_1 resp. α_2. It follows that the nodes of H' with labels α_1 and α_2 are adjacent (Lemma 4). We conclude that (H', ℓ') is a $(k-2, k)$-color-complete graph.

Next, consider the case that (H, ℓ) is a forest that satisfies the INP. Then H' is clearly a forest, since it can be obtained by contracting H (Lemma 4). If H contains no edges, then H' contains no edges, and trivially (H', ℓ') satisfies the INP. Note that this is the case if $|T| = k$. So it only remains to consider the case that H contains at least one edge, and therefore $|T| = k-1$. The last part of the proof is illustrated in Figure 4.

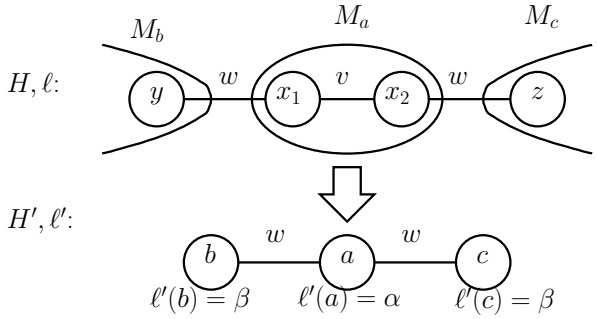


Fig. 4. An illustration of the proof of Lemma 15.

For every node $a \in V(H')$, denote by M_a the set of nodes of H that are contracted to obtain a, when constructing (H', ℓ') from (H, ℓ) as shown in Lemma 4 (so all nodes in M_a are labeled with a $G[T]$-coloring that is compatible with the $G[T \setminus \{v\}]$-coloring $\ell'(a)$). For every k-coloring α of $G[T \setminus \{v\}]$, there are at most two compatible colorings of $G[T]$, since $|T \setminus \{v\}| = k-2$. So since H satisfies the INP, the subgraph of H induced by the nodes that have an α-compatible label has maximum degree at most 1, and thus maximum component size at most 2. It follows that for every $a \in V(H')$, $|M_a| \leq 2$.

We now prove that (H', ℓ') satisfies the INP. Suppose to the contrary that H' contains a node a with $\ell'(a) = \alpha$, that has two neighbor nodes b and c with $\ell'(b) = \ell'(c) = \beta$. Let $w \in T \setminus \{v\}$ be the vertex on which α and β differ. So there are nodes $y \in M_b$ and $x_1 \in M_a$ that are adjacent in H, and nodes $z \in M_c$ and $x_2 \in M_a$ that are adjacent in H (Lemma 4). Because the adjacent colorings $\ell(y)$ and $\ell(x_1)$ differ on vertex w, they differ on no other vertex. The same holds for $\ell(z)$ and $\ell(x_2)$. As H satisfies the INP, it follows that $x_1 \neq x_2$, so $|M_a| \geq 2$, and thus $|M_a| = 2$.

We conclude that y, x_1, x_2, z is a path in H such that $\ell(y)(w) \neq \ell(x_1)(w)$, $\ell(x_1)(v) \neq \ell(x_2)(v)$, and $\ell(x_2)(w) \neq \ell(z)(w)$. Recall that labels of adjacent nodes in H differ on exactly one vertex. The colorings $\ell(y)$, $\ell(x_1)$, $\ell(x_2)$ and $\ell(z)$ all use $|T| = k-1$ different colors out of a total of k colors. Combining these facts shows that $\ell(y)(w) = \ell(x_2)(v) = \ell(z)(v)$. But since $\ell(y)$ and $\ell(z)$ are both compatible with β, $\ell(z)(w) = \ell(y)(w)$. This contradicts that $\ell(z)$ is a (proper) coloring of $G[T]$. We conclude that (H', ℓ') satisfies the INP. □

Next, we show that the introduce operation maintains the invariant.

Lemma 16 *Let $G = (V, E)$ be a $(k-2)$-connected k-colorable chordal graph and let T be a clique of G, with $T \neq V$, such that (G, T) can be obtained from $(G - v, T \setminus \{v\})$ using an introduce operation. If $\mathcal{C}_k^c(G - v, T \setminus \{v\})$ is a $(k-2, k)$-color complete graph, then $\mathcal{C}_k^c(G, T)$ is a $(k-1, k)$-color complete graph. If $|T| = k$ or $\mathcal{C}_k^c(G - v, T \setminus \{v\})$ is a forest that satisfies the INP, then $\mathcal{C}_k^c(G, T)$ is a forest that satisfies the INP.*

Proof. Let $(H, \ell) = \mathcal{C}_k^c(G - v, T \setminus \{v\})$ and $(H', \ell) = \mathcal{C}_k^c(G, T)$. We will use the fact that (H', ℓ') can be constructed from (H, ℓ) as shown in Lemma 5. By Lemma 13, $|T| \geq k-1$. If $|T| = k$, then H' is a set of isolated vertices, which proves the statement. So we may now assume that $|T| = k-1$.

First consider the case that (H, ℓ) is a $(k-2, k)$-color-complete graph. For every k-coloring α of $G[T]$, there exists exactly one node in H that has a label β that is compatible with α. So H' contains exactly one node with label α. Consider two colorings α_1 and α_2 of $G[T]$ that differ on exactly one vertex. If this vertex is v, then the nodes of H' with labels α_1 and α_2 are adjacent (Lemma 5). Otherwise, let $\beta_i = \alpha_i|_{T \setminus \{v\}}$ for $i = 1, 2$. The nodes with labels β_1 and β_2 are adjacent in H since it is a color-complete graph. Therefore, the nodes of H' with labels α_1 and α_2 are also adjacent in this case (Lemma 5). We conclude that (H', ℓ') is a $(k-1, k)$-color complete graph.

Next, consider the case that (H, ℓ) is a forest that satisfies the INP. From Lemma 5 it follows easily that (H', ℓ') satisfies the INP. We now prove that H' is a forest. Since $|T \setminus \{v\}| = k-2$, every node x of H has as label $\ell(x)$ a $(k-2)$-coloring of the complete graph $G[T \setminus \{v\}]$. So there are exactly two nodes in H' that correspond to x, which are adjacent (Lemma 5).

We will now show that for any edge $xy \in E(H)$, the following holds: if x_1 and x_2 are the vertices of H' that correspond to x, and y_1 and y_2 are the vertices of H' that correspond to y, then there is at most one edge in H' with one end in $\{x_1, x_2\}$ and one end in $\{y_1, y_2\}$. Observe that this property, combined with the fact that H contains no cycles, shows that H' contains no cycles.

Assume without loss of generality that x_1 and y_1 are adjacent in H'. Let $w \in T$ be the unique vertex with $\ell'(x_1)(w) \neq \ell'(y_1)(w)$ (note that $w \neq v$). Observe that the colorings $\ell'(x_1)$ and $\ell'(x_2)$ differ only on v, and that the same holds for the colorings $\ell'(y_1)$ and $\ell'(y_2)$. Since all colorings in ℓ' use $k-1$ colors out of k total colors, it follows that $\ell'(x_2)(v) = \ell'(y_1)(w) = \ell'(y_2)(w)$, and $\ell'(y_2)(v) = \ell'(x_1)(w) = \ell'(x_2)(w)$. Because all of these labels are (proper) colorings, we conclude that $\ell'(x_2)$ differs from the colorings $\ell'(y_1)$ and $\ell'(y_2)$ on both v and w, and $\ell'(y_2)$ differs from the colorings $\ell'(x_1)$ and $\ell'(x_2)$ on both v and w. Therefore, x_1y_1 is indeed the only edge between these two vertex groups. It follows that H' contains no cycles and is a forest. □

Finally, we show that the join operation maintains the invariant.

Lemma 17 *Let $G = (V, E)$ be a k-colorable chordal graph and let T be a clique of G, such that (G, T) can be obtained from (G_1, T) and (G_2, T) using a join operation. If one of $\mathcal{C}_k^c(G_1, T)$ or $\mathcal{C}_k^c(G_2, T)$ is an (m, k)-color complete graph, then $\mathcal{C}_k^c(G, T)$ equals the other graph. If both $\mathcal{C}_k^c(G_1, T)$ and $\mathcal{C}_k^c(G_2, T)$ are forests satisfying the INP, then $\mathcal{C}_k^c(G, T)$ is a forest satisfying the INP.*

Proof. Let $(H_1, \ell_1) = \mathcal{C}_k^c(G_1, T)$, $(H_2, \ell_2) = \mathcal{C}_k^c(G_2, T)$, and $(H, \ell) = \mathcal{C}_k^c(G, T)$. We use the fact that (H, ℓ) can be constructed from (H_1, ℓ_1) and (H_2, ℓ_2) as shown in Lemma 6.

First suppose that (H_1, ℓ_1) is a color-complete graph. Then Lemma 6 shows that every node of H_2 is combined with exactly one node of H_1 (there is exactly one node with the same label), so the nodes of H correspond bijectively to nodes of H_2. Furthermore, any edge of H_2 is maintained, since H_1 has edges between every pair of nodes labeled by colorings that differ on exactly one vertex. So (H, ℓ) equals (H_2, ℓ_2). If (H_2, ℓ_2) is a color-complete graph, the proof is analogous.

So it only remains to prove the statement in the case that both (H_1, ℓ_1) and (H_2, ℓ_2) are forests that satisfy the INP. From Lemma 6 it is easily seen that the INP is preserved in that case. We now argue that the resulting graph H is a forest. Suppose to the contrary that H contains a cycle $C = (u_0, v_0), (u_1, v_1), \ldots, (u_k, v_k)$ with $u_0 = u_k$ and $v_0 = v_k$ (we represent nodes of H by tuples (x, y) where $x \in V(H_1)$ and $y \in V(H_2)$, as shown in Lemma 6). Then $u_0, u_1, \ldots, u_k$ is a closed walk in H_1, and $v_0, v_1, \ldots, v_k$ is a closed walk in H_2, of length $k \geq 3$. Since H_2 is a forest, there is an index i such that $v_{i-1} = v_{i+1}$. It follows that $\ell_1(u_{i-1}) = \ell_2(v_{i-1}) = \ell_2(v_{i+1}) = \ell_1(u_{i+1})$. But u_{i-1} and u_{i+1} are both neighbors of u_i, so since H_1 satisfies the INP, $u_{i-1} = u_{i+1}$. We conclude that the vertices (u_{i-1}, v_{i-1}) and (u_{i+1}, v_{i+1}) in H are the same, contradicting that C is a cycle. So H is a forest that satisfies the INP. □

Combining the above lemmas yields:

Theorem 18 *Let $k \geq 3$. Let $G = (V, E)$ be a $(k-2)$-connected k-colorable chordal graph, and let $T \subseteq V(G)$ be a clique of G with $m = |T| \geq k-2$. Then $\mathcal{C}_k^c(G, T)$ is an (m, k)-color-complete graph, or it is a forest that satisfies the INP.*

Proof. We prove the statement by induction over $2|V| - |T|$. If $T = V(G)$, then $\mathcal{C}_k^c(G, T)$ is isomorphic to $\mathcal{C}_k(G)$, with the trivial label function (Lemma 3), so this is an (m, k)-color-complete graph (since T is a clique). Now assume that $T \neq V(G)$.

If (G, T) can be obtained from a graph $(G, T \cup \{v\})$ using a forget operation, where $T \cup \{v\}$ is a clique of G, then by induction, $\mathcal{C}_k^c(G, T \cup \{v\})$ is either an $(m+1, k)$-color-complete graph or a forest that satisfies the INP. Because $T \cup \{v\}$ is a clique on $m + 1$ vertices and G is k-colorable, $m \leq k - 1$. If $m = k - 1$ then $\mathcal{C}_k^c(G, T \cup \{v\})$ is a set of isolated nodes. This shows that Lemma 15 covers all cases, and thus $\mathcal{C}_k^c(G, T)$ satisfies the desired property.

If (G, T) can be obtained from a graph $(G - v, T \setminus \{v\})$ using an introduce operation then Lemma 13 shows that $G - v$ is $(k - 2)$-connected, and obviously it is chordal, so we may use induction to conclude that $\mathcal{C}_k^c(G - v, T \setminus \{v\})$ is either an $(m + 1, k)$-color-complete graph or a forest that satisfies the INP. Lemma 13 also shows that $|T| \geq k - 1$. This shows that Lemma 16 covers all cases, and thus $\mathcal{C}_k^c(G, T)$ satisfies the desired property.

In the remaining case, Lemma 11 shows that (G, T) is the join of two (smaller) graphs (G_1, T) and (G_2, T), which are $(k - 2)$-connected (Lemma 13), and chordal since they are induced subgraphs of G, so we can use the induction hypothesis. Then Lemma 17 can be applied, to show that $\mathcal{C}_k^c(G, T)$ satisfies the desired property. □

Remark 2. The examples in Figure 1 show that if we relax the connectivity requirement to $(k-3)$-connectedness, the property in Theorem 18 does not necessarily hold anymore: the examples in Figure 1(c) and (d) are not forests, and the example in Figure 1(e) does not satisfy the INP. Hence, we cannot generalize our polynomial-time result on $\mathcal{C}_k$-Reachability to $(k-3)$-connected chordal graphs in a straightforward way.

The characterization of $\mathcal{C}_k^c(G,T)$ in Theorem 18 does not yet guarantee that simply keeping track of the (relevant component of the) CSG yields a polynomial-time algorithm, as shown by the second example in Section 5. However, we will now show that it suffices to only keep track of the following essential information, which remains polynomially bounded.

6.4 An Efficient Algorithm: Computing Essential Information

Let $G=(V,E)$ be a graph with $T \subseteq V$, and let α and β be k-colorings of a supergraph of G. (The graph G should be viewed as a subgraph that occurs during the dynamic programming, while α and β are the colorings of the full graph.) Let $\alpha' = \alpha|_V$ and $\beta' = \beta|_V$. If $\mathcal{C}_k^c(G,T)$ is a forest with the α'-node x and β'-node y in the same component, then we define the *α-β-path* to be the unique path in $\mathcal{C}_k^c(G,T)$ with end vertices x and y (together with its vertex labels). Given the two colorings α and β, the *essential information* for $\mathcal{C}_k^c(G,T)$ consists of the following:

- whether the α' and β' nodes appear in the same component,
- whether $\mathcal{C}_k^c(G,T)$ is a forest, and
- in case the answers to both questions are yes: the α-β-path in $\mathcal{C}_k^c(G,T)$ (including vertex labels).

We also need to prove a polynomial upper bound on the length of the α-β-path. This is nontrivial, since the introduce operation may increase the length by a factor 2. However, we will show that this only happens when earlier, a forget operation has decreased the length by a similar amount. To formalize this, we use the following alternative length measure for paths in CSGs for recoloring.

For a subgraph F of G and $v \in V(G)$, we denote the neighbors of v in F by $N_F(v) = N(v) \cap V(F)$. Let (H,ℓ) be a labeled graph, where every node label $\ell(v)$ is a k-coloring of a complete graph on vertex set T. The set of colors *used by* a node $v \in V(H)$ is defined as $U(v) = \{\ell(v)(x) \mid x \in T\}$. If P is a subgraph in H and $v \in V(P)$, then the *node weight* for v is defined as $w_P(v) = |(\cup_{x \in N_P(v)} U(x)) \setminus U(v)|$. So this is the total number of colors that are used in the labels (colorings) for neighbors of v in P that are not used by the label for v itself. We define the *weight of a subgraph P of H* to be $w(P) = \sum_{v \in P} w_P(v)$. For example, consider the last CSG shown in Figure 2: the vertex with label 24 has weight 1 in the path with node labels $32, 34, 24, 23, 21$, but weight 2 in the path with node labels $32, 34, 24, 14, 12$. This weight depends on whether the corresponding path in the previous CSG (before forgetting f) contained the edge between nodes 124 and 324. The main idea is that for a path P, $w(P)$ bounds the length of P. This follows from the following simple observation, where we deduce amongst others that $w_P(v) \geq 1$ if v is not an isolated vertex.

Proposition 19 *Let (H,ℓ) be a labeled graph, where every node label $\ell(v)$ is a k-coloring of a complete graph on a vertex set T, such that adjacent nodes do not have the same label. Then for any subgraph P of H and any vertex $v \in V(P)$: $1 \leq w_P(v) \leq k - |T|$ if v is not an isolated vertex in P, and $w_P(v) = 0$ otherwise.*

We observe that, as soon as the α' and β' nodes are separated in some CSG that occurs during the dynamic programming, we may terminate and return NO.

Proposition 20 *Let $G' = (V', E')$ be a subgraph of $G = (V, E)$, and let α and β be two k-colorings of G. Let $\alpha' = \alpha|_{V'}$ and $\beta' = \beta|_{V'}$. For any $T' \subseteq V'$ and $T \subseteq V$: if the α' and β' nodes of $\mathcal{C}_k^c(G', T')$ are separated, then the α and β nodes of $\mathcal{C}_k^c(G, T)$ are separated.*

Proof. Suppose that the α and β nodes of $\mathcal{C}_k^c(G, T)$ are not separated. Then by Lemma 2, there exists a recoloring sequence $\gamma_0, \ldots, \gamma_p$ from α to β. Then restricting all of these colorings to V' yields a recoloring sequence $\gamma_0|_{V'}, \ldots, \gamma_p|_{V'}$ from α' to β' for G'. So using Lemma 2 again, the α' and β' nodes in $\mathcal{C}^c(G', T')$ are not separated. □

In our next three lemmas we consider a $(k-2)$-connected k-colorable graph G and we let α and β be two k-colorings of a supergraph of G (in line with the dynamic programming, where α and β are k-colorings of the input graph). Whenever we speak of α-β-paths in these lemmas, we formally mean the restriction of α and β to the vertex set of G (or some induced subgraph of G). Furthermore, in these lemmas, '*essentially polynomial time*' means polynomial in the entire input size, which includes the essential information; in particular, the path length. We will show later, namely in the proof of Theorem 24, that the maximum path length that can occur is at most $2(k+3)n$, which implies that our algorithm runs in polynomial time (in the usual sense).

Lemma 21 *Let G be a $(k-2)$-connected k-colorable chordal graph and let T be a clique of G with $k - 1 \leq |T|$, and $v \in T$. If we know the essential information for $\mathcal{C}_k^c(G, T)$, then in essentially polynomial time we can compute the essential information for $\mathcal{C}_k^c(G, T \setminus \{v\})$. If $\mathcal{C}_k^c(G, T)$ has a unique α-β-path P, then $\mathcal{C}_k^c(G, T \setminus \{v\})$ has a unique α-β-path P', and $w(P') \leq w(P)$.*

Proof. Theorem 18 shows that $\mathcal{C}_k^c(G, T)$ is either an (m, k)-color complete graph or a forest that satisfies the INP. Lemma 15 then shows that $\mathcal{C}_k^c(G, T \setminus \{v\})$ is a forest if and only if $\mathcal{C}_k^c(G, T)$ is a forest. Proposition 20 shows that if $\mathcal{C}_k^c(G, T)$ has no α-β-path, then $\mathcal{C}_k^c(G, T \setminus \{v\})$ has no α-β-path. If $\mathcal{C}_k^c(G, T)$ is a forest with a unique α-β-path P, then Lemma 4 shows that we can find an α-β-path P' in $\mathcal{C}_k^c(G, T \setminus \{v\})$ by starting with P, adjusting the labels, and possibly contracting some edges. This yields the unique α-β-path in the forest $\mathcal{C}_k^c(G, T \setminus \{v\})$.

We go more into detail on the construction of P' from P in order to prove that $w(P') \leq w(P)$. If $|T| = k$ then $\mathcal{C}_k^c(G, T)$ consists of only isolated nodes, and thus $\mathcal{C}_k^c(G, T \setminus \{v\})$ (which is a contraction of the former graph) as well, so the statement is trivial. So now assume that $|T| = k - 1$. By Proposition 19, every node in P has weight 1, and nodes in P' have weight at most 2. So to prove that $w(P') \leq w(P)$, it suffices to show that every node of P' with weight 2 results from contracting an edge of P (that is, contracting two nodes of weight 1). Denote by ℓ the node labels in $\mathcal{C}_k^c(G, T \setminus \{v\})$ (which are $(k-2)$-colorings of $G[T \setminus \{v\}]$). Consider a node $y \in V(P')$ with weight 2, so it has two neighbors $x, z \in V(P')$. Let $a \in U(x) \setminus U(y)$, and $b \in U(z) \setminus U(y)$. Since $w_{P'}(y) = 2$, it holds that $a \neq b$, so $U(x) \cup U(z) = \{1, \ldots, k\}$. So it is not possible to extend $\ell(x)$, $\ell(y)$ and $\ell(z)$ to k-colorings of $G[T]$ by assigning the same color to v, and therefore the node y resulted from contracting two nodes of P. □

Lemma 22 *Let $G = (V, E)$ be a $(k-2)$-connected k-colorable chordal graph. Let T be a clique of G with $T \neq V$, such that (G, T) can be obtained from $(G - v, T \setminus \{v\})$ by using an introduce*

operation. If we know the essential information for $\mathcal{C}_k^c(G-v, T\setminus\{v\})$, then in essentially polynomial time we can compute the essential information for $\mathcal{C}_k^c(G,T)$. If $\mathcal{C}_k^c(G,T)$ has a unique α-β-path P', then $w(P') \leq w(P)+2$ in the case where $\mathcal{C}_k^c(G-v, T\setminus\{v\})$ has a unique α-β-path P and $w(P')=0$ otherwise.

Proof. Let $(H,\ell) = \mathcal{C}_k^c(G-v, T\setminus\{v\})$ and $(H',\ell') = \mathcal{C}_k^c(G,T)$, such that (H',ℓ') is obtained from (H,ℓ) as shown in Lemma 5. Let $\alpha' = \alpha|_{V(G)}$ and $\beta' = \beta|_{V(G)}$. By Lemma 13 we find that $G-v$ is $(k-2)$-connected. Hence, we can use Theorem 18 to deduce that (H,ℓ) is either an (m,k)-color-complete graph (for $m = |T|-1$), or a forest that satisfies the INP.

By Lemma 13 we find that $|T| = k$ or $|T| = k-1$. First suppose that $|T| = k$. Then H' is a forest consisting of isolated nodes. So its α' and β' nodes are in the same component if and only if they are the same. This holds if and only if $\alpha'|_T = \beta'|_T$, and either (H,ℓ) is a $(k-1,k)$-color-complete graph or a forest with an α-β-path of length 0. Clearly the α-β-path in H' has length 0 in this case, and the label of its node is $\alpha|_T$. This shows that we can deduce, in essentially polynomial time, the essential information for H' if $|T| = k$.

Now suppose that $|T| = k-1$. Recall that (H,ℓ) is either a $(k-2,k)$-color-complete graph or a forest that satisfies the INP. Then by Lemma 16 the following holds. If (H,ℓ) is a $(k-2,k)$-color-complete graph, then (H',ℓ') is a $(k-1,k)$-color-complete graph. If H is a forest that satisfies the INP, then H' is a forest that satisfies the INP. Hence, H' is a forest if and only if H is a forest. Below we show how to deduce in essentially polynomial time the essential information for H'.

Proposition 20 shows that if H has no α-β-path, then H' has no α-β-path. Hence it remains to consider the case where H has a unique α-β-path P. We will apply Lemma 5 to the nodes of P to construct a (labeled) α-β-path P', which is a (labeled) subgraph of (H',ℓ'), and thus the unique α-β-path in (H',ℓ'), and show that $w(P') \leq w(P)+2$. (As an illustration of this proof, consider for instance how in Figure 2, the path P' with $w(P')=7$ between node labels 142 and 312 in the CSG with $T=(f,g,h)$ is deduced from the path P with $w(P)=5$ between node labels 14 and 31 in the previous CSG.)

Since $|T\setminus\{v\}| = k-2$, Lemma 5 shows that every node $x \in V(P)$ yields two adjacent nodes of H', which we will denote as x_1 and x_2, such that the labels $\ell'(x_1)$ and $\ell'(x_2)$ ($(k-1)$-colorings of $G[T]$) assign the two colors a and b that are not used by $\ell(x)$ to vertex v. For any two adjacent nodes x and y in P, $\ell(x)$ and $\ell(y)$ differ on exactly one vertex of $T\setminus\{v\}$, and both are $(k-2)$-colorings, so there is at least one color c that is used neither by $\ell(x)$ nor by $\ell(y)$. So we can choose indices $i,j \in \{1,2\}$ such that for the corresponding vertices x_i and y_j (as defined above) it holds that $\ell'(x_i)(v) = c$ and $\ell'(y_j)(v) = c$. Therefore x_i and y_j are adjacent in H' (Lemma 5). These two observations show that if we take the two nodes x_1 and x_2 for every $x \in V(P)$, then all of these nodes together induce a connected subgraph of H' that contains the α'-node and the β'-node of H'. Within this subgraph we can easily find the new α-β-path P'. Every node of the new path P' has weight 1 (Proposition 19). The total weight of P may increase by 2 if both end nodes of P are replaced by a pair of nodes this way. Nevertheless, we will now show that the weight cannot increase by more than 2, by showing that internal nodes y of P are only replaced by a pair of nodes y_1 and y_2 in P' if $w_P(y) = 2$.

Consider a node $y \in V(P)$ with neighbors x and z on P, such that without loss of generality the path P' contains the new nodes x_1, y_1, y_2, z_1, in this order. Let $u \in T\setminus\{v\}$ be the unique vertex that the colorings $\ell'(x_1)$ and $\ell'(y_1)$ differ on, and let $w \in T\setminus\{v\}$ be the unique vertex that the colorings $\ell'(y_2)$ and $\ell'(z_1)$ differ on. Since all of the colorings $\ell'(x_1), \ell'(y_1), \ell'(y_2), \ell'(z_1)$

use $k-1$ colors out of a total of k colors, we conclude that $\ell'(y_2)(v) = \ell'(x_1)(u)$, and similarly, $\ell'(y_1)(v) = \ell'(z_1)(w)$. It follows that the colorings $\ell(x) = \ell'(x_1)|_{T\setminus\{v\}}$ and $\ell(z) = \ell'(z_1)|_{T\setminus\{v\}}$ together still use all k colors, and therefore $w_P(y) = 2$. We conclude that internal nodes of P cannot contribute a weight increase, so $w(P') \leq w(P) + 2$. □

Lemma 23 *Let $G = (V, E)$ be a $(k-2)$-connected k-colorable chordal graph and let T be a clique of G, such that (G, T) can be obtained from (G_1, T) and (G_2, T) using a join operation. If we know the essential information for both $\mathcal{C}^c(G_1, T)$ and $\mathcal{C}^c(G_2, T)$, then in essentially polynomial time we can compute the essential information for $\mathcal{C}^c(G, T)$. If $\mathcal{C}^c_k(G, T)$ is a forest with a unique α-β-path P, then for at least one choice of $i \in \{1, 2\}$, $\mathcal{C}^c(G_i, T)$ is a forest with a unique α-β-path P_i, and $w(P) = w(P_i)$.*

Proof. Let $(H, \ell) = \mathcal{C}^c_k(G, T)$, $(H_1, \ell_1) = \mathcal{C}^c_k(G_1, T)$ and $(H_2, \ell_2) = \mathcal{C}^c_k(G_2, T)$ be labeled graphs such that (H, ℓ) is obtained from (H_1, ℓ_1) and (H_2, ℓ_2) as shown in Lemma 6. By Theorem 18, for $i \in \{1, 2\}$, (H_i, ℓ_i) is either a $(|T|, k)$-color complete graph or a forest that satisfies the INP (Lemma 13 shows that the graphs are $(k-2)$-connected).

By Lemma 17, H is a forest (that satisfies the INP) if and only if at least one of H_1 and H_2 is a forest. By Proposition 20, if there is no α-β-path in one of H_1 and H_2, then there is no α-β-path in H. So now assume that both H_1 and H_2 contain an α-β-path (though possibly not unique). If one of these, say H_i, is a forest with a unique α-β-path P_i, but the other is a color-complete graph, then the unique α-β-path P of H is the same as P_i (Lemma 17), and thus $w(P) = w(P_i)$.

It only remains to consider the case that both H_1 and H_2 are forests and contain a unique α-β-path; call these P_1 and P_2 respectively. If P_1 equals P_2, then H also has an α-β-path that equals these paths (Lemma 6), which is therefore the unique α-β-path P in H, with $w(P) = w(P_1) = w(P_2)$. We conclude the proof by showing the other direction. (This is similar to the last part of the proof of Lemma 17.) Suppose H has an α-β-path $P = v_0, \ldots, v_p$. Every node v_i of H corresponds to a pair v_i^1 and v_i^2 of nodes in H_1 resp. H_2, with $\ell(v_i) = \ell_1(v_i^1) = \ell_2(v_i^2)$, and $P_1 = v_0^1, \ldots, v_p^1$ and $P_2 = v_0^2, \ldots, v_p^2$ are α-β-walks in H_1 resp. H_2 (Lemma 6). If one of these, say P_1, is not a path, then since H_1 is a forest, there exists an index i such that $v_{i-1}^1 = v_{i+1}^1$. So $\ell(v_{i-1}) = \ell_1(v_{i-1}^1) = \ell_1(v_{i+1}^1) = \ell(v_{i+1})$. But since P is a path, v_{i-1} and v_{i+1} are distinct neighbors of v_i, so this contradicts the INP. We conclude that both P_1 and P_2 are paths, so P_1, P_2 and P are all equal, so $w(P) = w(P_1) = w(P_2)$. This concludes the proof, which shows that we can decide in essentially polynomial time whether H is a forest with an α-β-path, and compute it in that case. □

Combining the above statements yields the main result of this section:

Theorem 24 *Let G be a k-colorable $(k-2)$-connected chordal graph, and let α and β be two k-colorings of G. Then in polynomial time, we can decide whether $\mathcal{C}_k(G)$ contains an α-β path.*

Proof. Corollary 12 shows that for every chordal k-colorable graph G on n vertices, we can find in polynomial time a chordal nice tree decomposition on at most $(k+3)n$ nodes. So every node of this tree decomposition corresponds to a $(k-2)$-connected chordal subgraph H of G with terminal set T, such that either H is a clique with $T = V(H)$ (leaf nodes), or (H, T) can be obtained from the graph(s) corresponding to its child node(s) using a forget, introduce or join operation. (The fact that all of these graphs are $(k-2)$-connected follows inductively

using Lemma 13, and that they are chordal follows since they are induced subgraphs.) For every one of those terminal subgraphs, we compute the essential information, bottom up (Lemma 14, Lemmas 21–23). The computation terminates, answering NO, as soon as one subgraph (H, T) is encountered such that α and β are separated in $\mathcal{C}_k^c(H, T)$, which is correct by Proposition 20. (We remark that this can occur when (H, T) is obtained by a join operation, or by an introduce operation when $|T| = k$.) Otherwise, the computation terminates for the root node of the tree decomposition, which corresponds to the entire graph G itself, with some terminal set T, with the conclusion that either $\mathcal{C}_k^c(G, T)$ is a color-complete graph, or that it is a forest that contains an α-β-path. In either case, the answer to the problem is YES (Lemma 2).

Now we consider the complexity. We find the chordal nice tree decomposition in polynomial time, and it has at most $(k + 3)n$ nodes (Corollary 12). Computing the essential information for all nodes can be done in essentially polynomial time, that is, when the input size includes the α-β-path. However, every operation increases the weight of the path by at most 2 (Lemmas 21, 22 and 23), and in every case the weight of the path is an upper bound for its length (Proposition 19), so the maximum path length that can occur during the execution of the algorithm is at most $2(k + 3)n$. Together this shows that the whole procedure terminates in polynomial time. □

We stress that (m, k)-color complete graphs, which have $k!/(k - m)!$ nodes, are not computed explicitly in our algorithm. So indeed, in order to obtain a polynomial-time algorithm, we do not need to assume that k is a constant.

7 Discussion

Due to the PSPACE-completeness result of Hatanaka, Ito and Zhou [21] on $\mathcal{C}_k$-Reachability for chordal graphs, which holds when k is a sufficiently large constant, our polynomial-time algorithm cannot be extended to all chordal graphs. Since $\mathcal{C}_k$-Reachability problem is polynomial-time solvable for general graphs if $k = 3$ [15] and PSPACE-complete for $k = 4$ [8], it would nevertheless still be interesting to determine the complexity of $\mathcal{C}_4$-Reachability for chordal graphs (with at least one cut vertex). We refer to Remark 2 for a brief discussion on why our current proof technique does not work for this case. We also note that the complexity of $\mathcal{C}_4$-Reachability is open for proper interval graphs. Initial experimental results suggest that solving this problem is not straightforward.

Below we discuss a number of other possible directions for future work, which may require additional experimental results in order to refine our DP method. The two most important research goals are the following:

1. Explore for which other solution graph concepts $\mathcal{S}$ the DP method can be used to obtain polynomial-time algorithms for the $\mathcal{S}$-Reachability problem.

2. Explore which other commonly studied reconfiguration problems can be solved efficiently using CSGs.

The method of using CSGs can be applied to solve the $\mathcal{S}$-Connectivity problem. Hence, this problem (which is one of the central problems in reconfiguration) is a most suitable candidate problem for the second research goal. In this context we recall that the $\mathcal{C}_k$-Connectivity problem is trivial for chordal graphs [4] (see Section 1). Nevertheless, studying the complexity of the following related problem seems interesting. Call two k-colorings α and β of a graph G

compatible if they coincide on all k-cliques of G. Given a chordal graph G and k-coloring α, is the subgraph of $\mathcal{C}_k(G)$ induced by all k-colorings that are compatible with α connected?

Finally we discuss the list coloring generalization $\mathcal{C}_L$ of $\mathcal{C}_k$. In Remark 1, we explained how to generalize the DP rules presented in Section 4 to $\mathcal{C}_L$ (namely, by simply omitting all nodes that correspond to invalid vertex colors). In this way, we showed that the DP rules presented in [20] can be generalized. However, it is not obvious whether the results from Section 6 also generalize to list colorings. The following question by Hatanaka (asked at CoRe 2015) is also interesting: is there a polynomial-time algorithm for $\mathcal{C}_L$-Reachability restricted to trees? Note that $\mathcal{C}_k$-Reachability is trivial for trees, because $\mathcal{C}_k(G)$ is connected for every tree G and every integer $k \geq 3$ (see [4]; this also follows easily from Lemma 8).

Acknowledgement. The authors would like to thank Carl Feghali and Matthew Johnson for fruitful discussions and two anonymous reviewers for helpful comments.

References

1. H. Bodlaender, P. Bonsma, and D. Lokshtanov. The fine details of fast dynamic programming over tree decompositions. In *Proc. IPEC*, volume 8246 of *LNCS*, pages 41–53. Springer, 2013.
2. M. Bonamy and N. Bousquet. Recoloring graphs via tree decompositions. *European Journal of Combinatorics*, 69:200–213, 2018.
3. M. Bonamy, K. K. Dabrowski, C. Feghali, M. Johnson, and D. Paulusma. Recognizing graphs close to bipartite graphs with an application to colouring reconfiguration. *CoRR*, abs/1707.09817, 2017.
4. M. Bonamy, M. Johnson, I. Lignos, V. Patel, and D. Paulusma. Reconfiguration graphs for vertex colourings of chordal and chordal bipartite graphs. *Journal of Combinatorial Optimization*, 27:132–143, 2014.
5. P. Bonsma. The complexity of rerouting shortest paths. *Theoretical Computer Science*, 510:1–12, 2013.
6. P. Bonsma. Independent set reconfiguration in cographs and their generalizations. *Journal of Graph Theory*, 83(2):164–195, 2016.
7. P. Bonsma. Rerouting shortest paths in planar graphs. *Discrete Applied Mathematics*, 231:95–112, 2017.
8. P. Bonsma and L. Cereceda. Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theoretical Computer Science*, 410(50):5215–5226, 2009.
9. P. Bonsma, M. Kamiński, and M. Wrochna. Reconfiguring independent sets in claw-free graphs. In *Proc. SWAT 2014*, volume 8503 of *LNCS*, pages 86–97. Springer, 2014.
10. P. Bonsma, A. Mouawad, N. Nishimura, and V. Raman. The complexity of bounded length graph recoloring and CSP reconfiguration. In *Proc. IPEC 2014*, volume 8894 of *LNCS*, pages 110–121. Springer, 2014.
11. P. Bonsma and D. Paulusma. Using contracted solution graphs for solving reconfiguration problems. In *Proc. MFCS 2016,*, volume 58 of *LIPIcs*, pages 20:1–20:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
12. A. Brandstädt, V. Le, and J. Spinrad. *Graph Classes: A Survey*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, 1999.
13. L. Cereceda, J. van den Heuvel, and M. Johnson. Connectedness of the graph of vertex-colourings. *Discrete Mathematics*, 308(5):913–919, 2008.
14. L. Cereceda, J. van den Heuvel, and M. Johnson. Mixing 3-colourings in bipartite graphs. *European Journal of Combinatorics*, 30(7):1593–1606, 2009.
15. L. Cereceda, J. van den Heuvel, and M. Johnson. Finding paths between 3-colorings. *Journal of Graph Theory*, 67(1):69–82, 2011.
16. R. Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Berlin, 4th edition, 2010.
17. C. Feghali, M. Johnson, and D. Paulusma. A reconfigurations analogue of Brooks' theorem and its consequences. *Journal of Graph Theory*, 83(4):340–358, 2016.
18. P. Gopalan, P. G. Kolaitis, E. N. Maneva, and C. H. Papadimitriou. The connectivity of boolean satisfiability: Computational and structural dichotomies. *SIAM Journal on Computing*, 38(6):2330–2355, 2009.
19. A. Haddadan, T. Ito, A. E. Mouawad, N. Nishimura, H. Ono, A. Suzuki, and Y. Tebbal. The complexity of dominating set reconfiguration. *Theoretical Computer Science*, 651:37–49, 2016.

20. T. Hatanaka, T. Ito, and X. Zhou. The list coloring reconfiguration problem for bounded pathwidth graphs. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, 98(6):1168–1178, 2015.
21. T. Hatanaka, T. Ito, and X. Zhou. The coloring reconfiguration problem on specific graph classes. *Proc. COCOA 2017, Lecture Notes in Computer Science*, 10627:152–162, 2017.
22. T. Hatanaka, T. Ito, and X. Zhou. Parameterized complexity of the list coloring reconfiguration problem with graph parameters. *Theoretical Computer Science*, 739:65–79, 2018.
23. J. v. d. Heuvel. The complexity of change. In *Surveys in Combinatorics 2013*, pages 127–160. Cambridge University Press, 2013.
24. T. Ito, E. Demaine, N. Harvey, C. Papadimitriou, M. Sideri, R. Uehara, and Y. Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12–14):1054–1065, 2011.
25. T. Ito and E. D. Demaine. Approximability of the subset sum reconfiguration problem. *Journal of Combinatorial Optimization*, 28(3):639–654, 2014.
26. T. Ito, M. Kamiński, and E. Demaine. Reconfiguration of list edge-colorings in a graph. *Discrete Applied Mathematics*, 160(15):2199–2207, 2012.
27. T. Ito, K. Kawamura, H. Ono, and X. Zhou. Reconfiguration of list $L(2,1)$-labelings in a graph. *Theoretical Computer Science*, 544:84–97, 2014.
28. T. Ito, K. Kawamura, and X. Zhou. An improved sufficient condition for reconfiguration of list edge-colorings in a tree. *IEICE TRANSACTIONS on Information and Systems*, 95(3):737–745, 2012.
29. M. Johnson, D. Kratsch, S. Kratsch, V. Patel, and D. Paulusma. Finding shortest paths between graph colourings. *Algorithmica*, 75(2):295–321, 2016.
30. M. Kamiński, P. Medvedev, and M. Milanič. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439:9–15, 2012.
31. T. Kloks. *Treewidth: computations and approximations*, volume 842 of *LNCS*. Springer, 1994.
32. F. G. König, M. E. Lübbecke, R. H. Möhring, G. Schäfer, and I. Spenke. Solutions to real-world instances of pspace-complete stacking. In *Proc. ESA 2007*, volume 4698 of *Lecture Notes in Computer Science*, pages 729–740, 2007.
33. D. Lokshtanov, A. E. Mouawad, F. Panolan, M. S. Ramanujan, and S. Saurabh. Reconfiguration on sparse graphs. *Journal of Computer and System Sciences*, 95:122–131, 2018.
34. A. E. Mouawad, N. Nishimura, V. Pathak, and V. Raman. Shortest reconfiguration paths in the solution space of boolean formulas. *SIAM Journal on Discrete Mathematics*, 31(3):2185–2200, 2017.
35. A. E. Mouawad, N. Nishimura, V. Raman, and S. Siebertz. Vertex cover reconfiguration and beyond. *Algorithms*, 11(2):20, 2018.
36. A. E. Mouawad, N. Nishimura, V. Raman, N. Simjour, and A. Suzuki. On the parameterized complexity of reconfiguration problems. *Algorithmica*, 78(1):274–297, 2017.
37. A. E. Mouawad, N. Nishimura, V. Raman, and M. Wrochna. Reconfiguration over tree decompositions. In *Proc. IPEC 2014*, volume 8894 of *LNCS*, pages 246–257. Springer, 2014.
38. R. Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.
39. N. Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018.
40. F. S. Roberts. Indifference graphs. *Proof Techniques in Graph Theory,* Academic Press, New York, pages 139–146, 1969.
41. M. Wrochna. Homomorphism reconfiguration via homotopy. In *Proc. STACS 2015*, volume 30 of *LIPIcs*, pages 730–742. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.
42. M. Wrochna. Reconfiguration in bounded bandwidth and tree-depth. *Journal of Computer and System Sciences*, 93:1–10, 2018.
43. T. v. d. Zanden. Parameterized complexity of graph constraint logic. In *Proc. IPEC 2015*, volume 43 of *LIPIcs*, pages 282–293. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.